Departamento de Ingeniería de Sistemas y Automática

PROYECTO FIN DE CARRERA

IMPLEMENTACIÓN DE APLICACIONES "SERIOUS GAMES" PARA LA EVALUACIÓN DE DESTREZA MANUAL

Autor: Cynthia Mª Cubeiro Tuimil

Tutor: Alberto Jardón Huete

Leganés, octubre de 2016

Título: IMPLEMENTACIÓN DE APLICACIONES "SERIOUS GAMES" PARA LA EVALUACIÓN DE DESTREZA MANUAL

Autor: Cynthia Mª Cubeiro Tuimil

Director:

EL TRIBUNAL

Presidente: Francisco José Rodríguez Urbano

Vocal: María Fernández Álvarez

Secretario: Pablo Zumel Vaquero

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 28 de octubre de 2016 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de sobresaliente (10).

VOCAL

SECRETARIO PRESIDENTE

RESUMEN

El *serious game* realizado para este proyecto nace de la necesidad del Laboratorio de Análisis del Movimiento, Biomecánica, Ergonomía y Control Motor (LAMBECOM) [1] de contar con un videojuego para la rehabilitación física de sus pacientes donde se trabaje la mejora de la disociación de los dedos y de la coordinación bimanual a la vez que se extraigan los datos relevantes de los ejercicios.

Para ello se ha diseñado un videojuego que simula un piano de diez teclas en el que cada tecla corresponde a uno de los dedos de la mano y tendrán que ir pulsándose en el orden que el juego vaya indicando. La programación del videojuego se ha realizado utilizando el motor de videojuegos de código abierto Unity. Por otro lado, la monitorización de la mano se realiza a través del dispositivo Leap Motion, siendo su software capaz de reconocer cada dedo de la mano y sus respectivos huesos y extraer la información necesaria para la futura evaluación de los fisioterapeutas.

Palabras clave: Serious game, rehabilitación con videojuegos, realidad virtual, Leap Motion

ABSTRACT

The serious game created for this project started with the need of the Laboratory of Analysis of Movement, Biomechanics, Ergonomics and Motor Control (LAMBECOM) [1] to have a video game for rehabilitation patients to help improve the dissociation of the fingers and the bimanual coordination while the relevant data is extracted from the performed exercises.

To that end, a video game that simulates a ten key piano was designed where each key corresponds to one of the fingers and they must be pressed in the order the game indicates. The video game programming was created using the opensource game engine Unity. On the other hand, hand monitoring is done through the device known as Leap Motion. The Leap Motion's software is able to recognize each finger and their bones and extract the necessary information for future evaluation of the physiotherapists.

Keywords: serious game, rehabilitation with video games, virtual reality, Leap Motion

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1.

INTRODUCCIÓN

1.1. INTRODUCCIÓN

El mundo de los videojuegos está creciendo a pasos agigantados en la sociedad tecnológica en la que nos encontramos. El realismo de los gráficos y la capacidad de interacción con el juego que existe en la actualidad nada tienen que ver con los de hace 20 años y, no solo eso, sino que se encuentra en pleno auge.

Sin embargo, los videojuegos en la actualidad no se utilizan únicamente con el objetivo de divertir al usuario, sino que de esta evolución han surgido los "juegos serios" o "juegos formativos" (del inglés "*serious games*"). Estos juegos hacen referencia a cualquier juego que tenga una finalidad distinta a la de jugar por diversión, es decir, que busquen enseñar o entrenar, contribuyendo a la adquisición de habilidades y conocimientos que se pueden aplicar a las actividades diarias y se utilizan para hacer frente a los retos normales de la vida [2].

De la vinculación entre los *serious games* y el desarrollo de los controladores que permiten la detección de los movimientos del cuerpo humano, nace la rehabilitación con videojuegos. Este tipo de terapia se beneficia de la interacción humana con la realidad virtual para realizar ejercicios de manera más divertida y motivadora, pero además, permite la monitorización de las partes del cuerpo que se requiera y se puede obtener información que la simple observación humana no alcanza.

Para este proyecto se ha creado un videojuego que servirá de herramienta en el proceso de rehabilitación de manos, utilizando un sistema de captura de movimientos y un software de código abierto, para pacientes que hayan sufrido lesiones neurológicas o físicas que afecten a las extremidades superiores, en particular a las manos y dedos.

1.2. MOTIVACIÓN

Este videojuego pertenece al proyecto RoboHealth [3], cuyo objetivo es el desarrollo de robots de asistencia y rehabilitación para personas con movilidad limitada, habilidades cognitivas reducidas o enfermedades crónicas, con la finalidad de ayudarles a obtener una mayor calidad de vida aportándoles independencia. A la necesidad del Laboratorio de Análisis del Movimiento, Biomecánica, Ergonomía y Control Motor (LAMBECOM) de contar con una herramienta de trabajo que mejorase la calidad de la rehabilitación de manos y su evaluación, RoboHealth aporta este *serious game* como solución.

Con este trabajo se pretende crear una alternativa o complemento a la rehabilitación física tradicional, con el valor añadido de que el paciente se entretenga y pueda llegar a olvidar que está realizando unos ejercicios por obligación. Pero sobre todo es importante resaltar que lo que aporta incluir sistemas de captura de movimientos en la rehabilitación motora es poder llegar donde el ojo humano no alcanza y facilitar información precisa sobre el esqueleto humano, sus articulaciones y sus respectivos movimientos para ser analizada posteriormente por el terapeuta.

En este proyecto se expone la elección de las herramientas adecuadas para el desarrollo del *serious game*, escogiendo entre la última tecnología disponible en el mercado con el principal objetivo de crear un videojuego asequible para poder ser utilizado en cualquier centro médico o incluso por los pacientes en su propia casa. Esto se cumplirá componiéndolo de hardware económico y software de código abierto.

1.3. OBJETIVOS

Como se mencionó previamente, el objetivo principal de este proyecto es la creación de un "*serious game*" enfocado a la rehabilitación de manos que mejore la calidad del tratamiento impartido por los terapeutas y que aporte información adicional para la evaluación de los resultados y el progreso del paciente.

De cara a poder mejorar la calidad de la terapia y que se pueda llegar a más pacientes, debe tratarse de un juego sencillo, fácil de entender y manejar, pero a su vez deberá contar con un diseño entretenido de modo que el paciente se sienta motivado a la hora de realizar sus ejercicios.

Como también se señalaba en el apartado anterior, otro de los objetivos de la solución propuesta en este proyecto es que pueda ser utilizado tanto en centros médicos como desde casa para facilitar a los pacientes que puedan aumentar su tiempo de ejercitación evitando desplazamientos a las grandes instalaciones. Para que esto pueda cumplirse, una característica fundamental es que sea económico y con manipuladores de pequeñas dimensiones.

Tratará de ser lo menos excluyente posible con el público objetivo, es decir, se podrá particularizar teniendo en cuenta las lesiones y estado físico de cada paciente. En este caso se incluirá rehabilitación para pacientes con lesiones tanto neurológicas como físicas pero, además, podrá ser utilizado por personas en sillas de ruedas, por personas con problemas de audición, ancianos y niños sin conocimientos de tecnología, entre otros, cumpliendo lo mayormente posible con los paradigmas de Diseño para todos [4].

Para hacer aún más personalizado el videojuego, se utilizará software de código abierto. Esto significa que cualquier persona con acceso a la aplicación y conocimientos de programación podrá manipular el programa con la finalidad de personalizarlo y realizar mejoras al juego.

Por último, uno los objetivos más significativos es el que se consigue añadiendo un sistema de captura de movimientos al videojuego. Este sistema rastreará los huesos y articulaciones de la mano humana de manera que permitirá una evaluación más precisa de los movimientos del paciente. Los resultados que se obtengan de este reconocimiento serán guardados en un archivo facilitando la evaluación por parte de los terapeutas de cada sesión y de los avances del paciente tanto si los ejercicios se realizan en sus instalaciones como desde casa.

1.4. ESTRUCTURA DE LA MEMORIA

Para facilitar la lectura de la memoria, se incluye a continuación una breve descripción de lo que se hablará en cada capítulo.

En el capítulo dos se contextualizará la terapia de rehabilitación con videojuegos y cómo surgió. Para ello se describirá cómo es la rehabilitación física tradicional, los tipos de sistemas de captura de movimientos existentes y cómo a partir de la combinación de ambos surge la rehabilitación con videojuegos. Por último, se pondrá al día de cómo el Leap Motion (dispositivo utilizado en este proyecto) está siendo usado dentro de los avances tecnológicos en el campo de la medicina.

Los medios empleados para la creación del *serious game* se detallarán en el capítulo tres. En él se incluirá un apartado para el software y otro para el hardware y, puesto que para la elección de ellos se han realizado sus correspondientes comparativas con otros productos del mercado, estas se incorporarán en cada uno.

A continuación, en el capítulo cuatro, se explicará en qué consiste el *serious game* que se ha realizado para este proyecto y todo su desarrollo, explicando sus especificaciones, trabajos previos y programación del mismo.

Finalmente, en el capítulo cinco, se expondrán las conclusiones extraídas tras la realización del proyecto y los posibles trabajos futuros que presenten una continuación a este.

Capítulo 2.

CONTEXTO DE LA REHABILITACIÓN CON VIDEOJUEGOS

En este proyecto se aborda el desarrollo de un tipo de *serious game* en particular: los videojuegos para la terapia de rehabilitación física. A continuación se pone en contexto dicha terapia exponiendo en los siguientes apartados la rehabilitación tradicional, los tipos de sistemas de captura de movimientos existentes, cómo surgió la rehabilitación con videojuegos y, por último, se introduce en qué ámbitos de la medicina se utiliza hoy en día el Leap Motion y su aplicación.

2.1 LA REHABILITACIÓN TRADICIONAL

Tradicionalmente, la rehabilitación física y su correspondiente evaluación en pacientes se lleva a cabo a través de las indicaciones y supervisión de fisioterapeutas. La terapia de rehabilitación motora se realiza en pacientes que han sufrido algún trastorno o lesión neurológica o musculoesquelética con la finalidad de restaurar su habilidad para realizar con independencia las actividades básicas de la vida diaria (ABVD) o recuperar una función perdida o disminuida realizando ejercicios en una base regular.

Con el fin de realizar estudios más fiables y objetivos, se utilizan métodos estandarizados que pueden ayudar a los fisioterapeutas u otros profesionales de la salud durante el curso del tratamiento a [5]:

- Priorizar las intervenciones en el tratamiento basado en déficits motrices y sensoriales específicos e identificables.
- Crear metas apropiadas a corto y largo plazo para el tratamiento basado en el resultado de las escalas, su experiencia profesional y los deseos del paciente.

- Evaluar la carga potencial de los cuidados y monitorizar cualquier cambio basado tanto en mejoras como en disminución de puntuaciones.

Una escala es un instrumento de trabajo en el que una serie de propiedades que interesa evaluar en pacientes se miden cuantitativa (mucho, bastante, poco, nada) o cualitativamente (siempre, casi siempre, a veces, casi nunca, nunca) [6].

Lo más importante de una escala es que sea:

- Válida, tanto en el contenido como en el criterio.
- Fiable.
- Reproducible, es decir, que se pueda dar en la misma secuencia.
- Sensible, es decir, que detecte los pequeños cambios.

Una de las escalas más completas para valorar la funcionalidad de miembros superiores es el *Fugl Meyer Assessment*, pero si lo que se desea es valorar la destreza, coordinación y manipulación con las manos, se utilizan, entre otras, las siguientes escalas: *Box and Block Test* (BBT), *Nine Hole Peg Test* (NHPT) y *Purdue Pegboard Test* (PPT). Estos métodos miden la destreza manual (en el caso del BBT), de los dedos (en el caso del NHPT) y de la punta de los dedos (en el caso del PPT) y se trata de pruebas rápidas, sencillas y económicas que se pueden utilizar con una amplia variedad de pacientes [7]. A continuación se expone con más profundidad cada una de estas escalas.

2.1.1. FUGL-MEYER ASSESSMENT

El *Fugl-Meyer Assessment* (FMA) [7] es específica para pacientes que han sufrido un derrame cerebral y mide el índice de discapacidad basándose en el desempeño del paciente. Se trata de una evaluación larga de llevar a cabo puesto que puede durar entre 34 y 110 minutos si se realiza entera, por ello suele gestionarse por separado dividiéndola en las siguientes calificaciones: funcionamiento motriz, equilibrio, sentido de la sensación y funcionamiento articular [8]. Se analiza el movimiento, la coordinación y reflejos de hombros, codos, antebrazos, muñecas, manos, cadera,

rodillas y tobillos y se aplica clínicamente para determinar la gravedad de la enfermedad, describir la recuperación motriz y para planificar y evaluar el tratamiento.

NOMBRE: _____ FECHA:_____

FORMATO DE REGISTRO: **ESCALA DE FUGL – MEYER**

MIEMBRO SUPERIOR						
A HOMBRO/CODO/ANTEBRAZO				**B** MUÑECA		
I	Reflejos	Flexores		Codo 90°	Estabilidad	
		Extensores		Codo 90°	Flexo-extensión	
II a	Hombro	Retracción		Codo 0°	Estabilidad	
		Elevación		Codo 0°	Flexo-extensión	
		Abducción			Circunducción	
		Rotación externa			**SUBTOTAL**	
	Codo	Flexión		**C** MANO		
	Antebrazo	Supinación		Flexión en masa		
b	Hombro	Aducción – rotación interna		Extensión en masa		
	Codo	Extensión		Prensión A	Extensión MCF, flexión IFP, P	
Cs*	Antebrazo	Pronación		Prensión B	Aducción del pulgar	
III	Mano a columna lumbar			Prensión C	Pinza 1-2	
	Hombro	Flexión de 0° – 90°		Prensión D	Cilindro	
	Codo 90°	Prono - supinación		Prensión E	Esfera	
IV	Hombro	Abducción de 0° – 90°			**SUBTOTAL**	
Ss**		Flexión de 90° – 180°		**D** COORDINACIÓN/VELOCIDAD		
	Codo 0°	Prono - supinación		Temblor		
V	Actividad refleja			Dismetría		
				Velocidad		
		SUBTOTAL		**SUBTOTAL**		

Cs* = con sinergia, Ss** = sin sinergia, MCF = Articulaciones metacarpofalángicas, FP = Articulaciones interfalángicas proximales, P = Pulgar

TOTAL: _____

Tabla 1: Formato de registro de la escala de Fugl-Meyer.

2.1.2. BOX AND BLOCK TEST

Se trata de un ejercicio en el que el paciente se sienta frente a una caja con dos compartimentos divididos por una partición, donde en uno de ellos se encuentran 150 cubos de madera. El paciente deberá durante un minuto transportar uno a uno el mayor número de cubos posible por encima de la partición y soltarlos en el compartimento de al lado. El ejercicio se realizará primeramente con la mano dominante y al finalizar el test el examinador contará los cubos, resultando una puntuación igual al número de cubos desplazados. Tras ello, se repetirá el test con la mano no dominante, puntuando cada mano por separado.

Figura 1: Caja del Box and Block Test.

El número de cubos desplazados permite medir la destreza manual durante la rehabilitación, indicando cuantos más cubos mayor destreza. Los resultados pueden ser comparados con valores referenciados a tests realizados con prótesis o por sujetos sanos. Las instrucciones completas se pueden revisar en [9].

2.1.3. NINE HOLE PEG TEST

El paciente deberá ir cogiendo una a una las varillas que aparecen en el recipiente de la figura 2 e introduciéndolas en cada uno de los nueve agujeros en cualquier orden hasta rellenarlos todos. Al finalizar, deberá recogerlos uno a uno y devolverlos al recipiente de nuevo. El examinador comenzará a cronometrar el tiempo en el momento en el que el paciente toque la primera varilla y lo parará al introducir la última varilla en el recipiente. El ejercicio se realizará primero con la mano dominante y se repetirá después con la no dominante.

Figura 2: Caja del Nine Hole Peg Test.

El NHPT mide la destreza de los dedos con una alta fiabilidad y sensibilidad para detectar pequeñas deficiencias en la actividad de la mano.

2.1.4. PURDUE PEGBOARD TEST

Es un test neuropsicológico que mide la destreza manual y coordinación bimanual. El test implica dos habilidades distintas: movimientos amplios de brazos, manos y dedos y destreza en la yema de los dedos. Un mal desempeño del ejercicio es señal de déficits en movimientos complejos, visualmente guiados o coordinados que están probablemente mediados por circuitos relacionados con los ganglios basales [10].

Figura 3: Tablero del Purdue Pegboard Test.

En el tablón se encuentran dos filas con 25 agujeros cada una y cuatro recipientes con el material necesario. El paciente deberá introducir las pequeñas varillas metálicas en cada agujero durante 30 segundos, primero con una mano, después con la otra y finalmente con las dos.

Los resultados de un análisis de correlación sugieren que la capacidad de una persona en el PPT es un buen indicador de su habilidad para utilizar un móvil en clima frío y se utiliza también para contratar a los empleados más cualificados en las empresas de cadenas de montaje.

2.2 SISTEMAS DE CAPTURA DE MOVIMIENTOS

La captura de movimientos humanos para rehabilitación ha sido un tema activo de investigación desde los años 80, provocado por el aumento del número de pacientes que han sufrido un ictus u otra discapacidad motora.

En 1973, Johansson estudió su famoso experimento psicológico Moving Light Display (MLD) para percibir el movimiento biológico. Adjuntó pequeños marcadores reflectantes en las articulaciones de sujetos humanos, lo que permitió que estos marcadores fuesen monitorizados durante su trayectoria. Este experimento se convirtió en un hito del rastreo del movimiento humano [11].

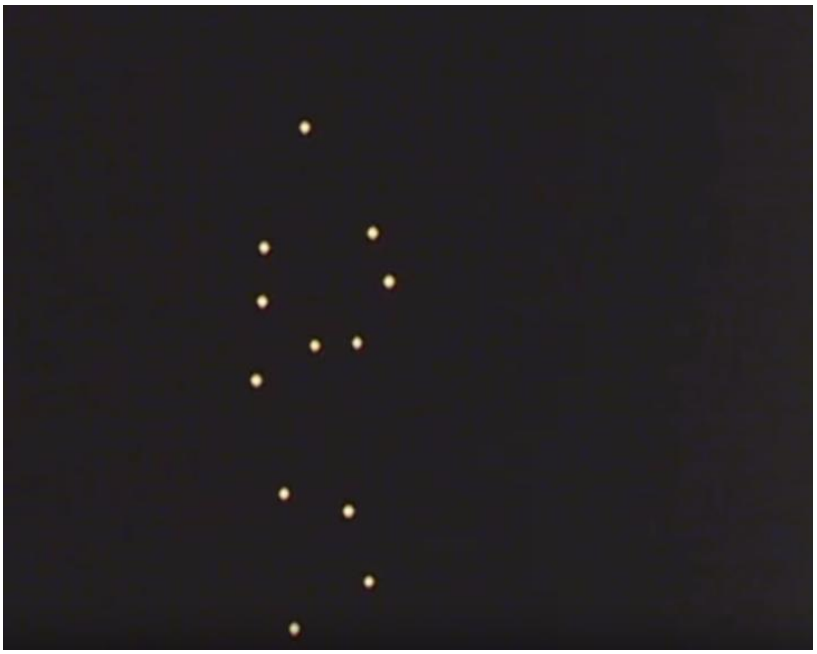


Figura 4: Persona caminando con Moving Light Display.

Figura 5: Persona haciendo flexiones con Moving Light Display

En la figura 4 aparece una persona caminando de frente y en la imagen 5 se muestra una persona haciendo flexiones mientras llevan puestos los marcadores reflectantes.

En este estudio se utilizó en particular un rastreo de movimientos a partir de marcadores adjuntos al cuerpo, pero existe una gran variedad de sensores para este fin. En la tabla 2 se muestran los distintos tipos de sistemas de captura de movimiento (MoCap) que existen, ninguno de ellos se creó con la finalidad de ser utilizados para rehabilitación motora, pero son mayormente los optoelectrónicos los que hoy en día forman parte de los sistemas de rastreo de movimiento de la terapia basada en videojuegos.

MoCap No optoelectrónicos	MoCap Optoelectrónicos
	(i) Trigonometría con marcadores
(i) Sensores de inercia	y cámaras infrarrojas
(ii) Sistemas magnéticos	(ii) Basadas en contraste
(iii) Sistemas vestibles	(a) Con marcadores de colores
(iv) Sistemas mecánicos	(b) Con detección de piel
	(iii) Basadas en la profundidad

Tabla 2: Tipos de sistemas de captura de movimiento [12].

Sistemas	Precisión	Compactibilidad	Cómputo	Coste	Incovenientes
De inercia	Alta	Alta	Eficiente	Bajo	Desvíos
Magnéticos	Media	Alta	Eficiente	Bajo	Materiales ferromagnéticos
Ultrasonidos	Media	Baja	Eficiente	Bajo	Oclusión
Guantes	Alta	Alta	Eficiente	Medio	Postura parcial
Con marcadores	Alta	Baja	Ineficiente	Medio	Oclusión
Sin marcadores	Alta	Alta	Ineficiente	Bajo	Oclusión
Combinados	Alta	Baja	Ineficiente	Alto	Multidisciplinario
Robots	Alta	Baja	Ineficiente	Alto	Moción limitada

Tabla 3: Comparación del comportamiento de los distintos sistemas de rastreo de movimiento [11].

Los sensores optoelectrónicos[12], como podemos observar en la parte derecha de la tabla 2, rastrean los movimientos haciendo uso de un escaneo basado o bien en el contraste o bien en la profundidad. Los sistemas de percepción de colores, en ingeniería de rehabilitación, pueden rastrear marcadores de un color específico que se

encuentran adjuntos al cuerpo y que representan cada segmento del esqueleto y las articulaciones para obtener su posición y orientación, o pueden rastrear el color de la piel del paciente. Por otro lado, los sistemas basados en profundidad segmentan y rastrean el cuerpo humano por las secuencias de la profundidad de las imágenes. Este último tipo de escaneo aporta la importante característica de rastrear el esqueleto y lo aportan dispositivos como el Microsoft Kinect o el Leap Motion. Ambos proporcionan un kit de desarrollo de software (SDK) que da acceso a los desarrolladores a las posiciones y orientaciones de las articulaciones del cuerpo, en el caso de la Kinect, y de las articulaciones de la mano en el caso del Leap Motion. Una imagen en profundidad contiene información relacionada con la distancia entre las superficies del objeto 3D y la cámara y con esta información se hace considerablemente más fácil y preciso segmentar.

La tecnología sensorial y computacional que puede ser utilizada para la captura de movimientos ha avanzado drásticamente en los últimos años, volviéndose más cualificados y asequibles y registrando datos sobre la cinemática del cuerpo humano con alta precisión y fiabilidad. Esto ha liderado a una explosión en nuevas tecnologías, especialmente en dispositivos de bajo coste para videojuegos basados en sistemas de rastreo óptico, frecuencias de radio, cámaras infrarrojas y tecnología táctil que se han vuelto accesibles para casi todo el mundo [13].

La rehabilitación física es un proceso dinámico que utiliza instalaciones disponibles para corregir cualquier movimiento indeseado en el comportamiento con el fin de alcanzar una expectación (por ejemplo la postura ideal). Por tanto, durante el transcurso de la terapia el movimiento de los pacientes necesita ser continuamente monitorizado y rectificado para mantener un patrón de moción correcto. En consecuencia, detectar los movimientos humanos resulta vital y necesario en el diseño de videojuegos enfocados a la rehabilitación.

2.3 REHABILITACIÓN CON VIDEOJUEGOS

Los ejercicios efectuados durante las terapias de rehabilitación física tienen en común que se realizan en grandes instalaciones que no son siempre accesibles para todos los pacientes, son supervisadas bajo la subjetividad del terapeuta encargado de llevarlas a cabo y pueden resultar pesadas a quien las debe realizar frecuentemente para hacer posible su recuperación. Es por ello que desde hace unos 15 años se viene aplicando la tecnología de captura de los movimientos humanos a través de la realidad virtual en prácticas de rehabilitación [14]. Son muchas las ventajas que aporta la terapia basada en videojuegos frente a la terapia convencional y se exponen a continuación.

Entre las principales ventajas encontramos la de aportar especificidad y adaptabilidad a cada paciente y enfermedad debido a que el entorno artificial puede ser fácilmente modificado, permitiendo así un diseño óptimo para una terapia individualizada. Los ejercicios de la terapia requieren de una repetitividad que es fácilmente proporcionada a través de un videojuego y, además, se pueden llevar a cabo más repeticiones en cada sesión ya que se envían los estímulos al paciente con mayor rapidez. Una de las características más significativas de esta tecnología es que tiene la habilidad de enganchar al paciente puesto que aporta estímulos ricos y funcionales y un contexto motivador, aumentando así la participación activa del sujeto en su programa de rehabilitación física. Para solventar el problema de la subjetividad de la terapia convencional que mencionábamos anteriormente, la tecnología aporta la capacidad de una evaluación precisa a través de la captura de movimientos del cuerpo que se detalló en el apartado 2.2., liberando al terapeuta para involucrarse más en guiar físicamente al paciente. También se hacía referencia a la necesidad de grandes instalaciones donde llevar a cabo las terapias de rehabilitación motora donde se albergan equipos grandes, cualificados y caros, pero a las que no todos los pacientes pueden desplazarse con la frecuencia que se requiere, mientras que la rehabilitación basada en videojuegos da paso a la tele-rehabilitación, es decir, los usuarios pueden rehabilitarse cómodamente en su casa y los terapeutas pueden acceder a los resultados de la sesión correspondiente a través de un acceso remoto para su

evaluación. Por último, esta forma de terapia propuesta garantiza la seguridad necesaria para los pacientes en rehabilitación física.

No obstante, ¿son igual de efectivos ambos tipos de terapias? A lo largo del tiempo se vienen realizando estudios que lo analizan. Se ha demostrado la eficacia de adicionar a la rehabilitación tradicional sesiones de terapia basada en videojuegos [15] puesto que se extiende el tiempo dedicado a ejercitar, con la ventaja de que los juegos ofrecen el potencial de enganchar al paciente hasta el punto de no enfocarse en el hecho de que están en una sesión de rehabilitación. Pero también se ha demostrado la viabilidad de la terapia realizada a través de realidad virtual [16] y que es igual de efectiva en comparación con la terapia estándar [12], [14].

En estos videojuegos se utiliza la realidad virtual, en la que los usuarios interactúan con las imágenes que se muestran, moviendo y manipulando objetos virtuales y desempeñando otras acciones de una manera que trata de sumergirlos en un ambiente simulado de modo que se genera una sensación de presencia en el mundo virtual. Se usa una cámara de video y un software para rastrear los movimientos humanos, sin necesidad de colocar marcadores en lugares específicos del cuerpo. Para poder llevar a cabo esta interacción se utilizan dispositivos de captura de movimiento humano que permiten la monitorización necesaria de los movimientos del sujeto.

Dispositivos como la Kinect, Nintendo Wii, Sony X-Box, EyeToy o el Leap Motion, son algunos de los sistemas de captura óptica que, además de ser populares entre los videojuegos, están siendo utilizados como detectores en rehabilitación física.

El brazo Rutgers es uno de los primeros prototipos compuestos por un PC, un sistema de captura de movimientos y una mesa de baja fricción para la rehabilitación de las extremidades superiores. El sistema fue testado en pacientes con ictus crónico y demostraron mejoras en el control motor del brazo y el rango de movimiento del brazo (calificación de Fugl-Meyer test) [13].

Figura 6: Paciente con ictus utilizando el brazo Rutgers [17].

Hoy en día existen sistemas más modernos como VirtualRehab [18]. Se trata de un sistema de rehabilitación física clínicamente validado que usa la tecnología de videojuegos y permite la monitorización del progreso de los pacientes desde cualquier lugar del mundo. Se trata del primer software de rehabilitación virtual en ser clasificado como un producto sanitario, consiguiendo la marca CE cumpliendo con el sistema regulatorio de la directiva sobre productos sanitarios. Como se puede observar en la figura 7, utilizan la Kinect (imagen de la izquierda) y el Leap Motion (derecha).

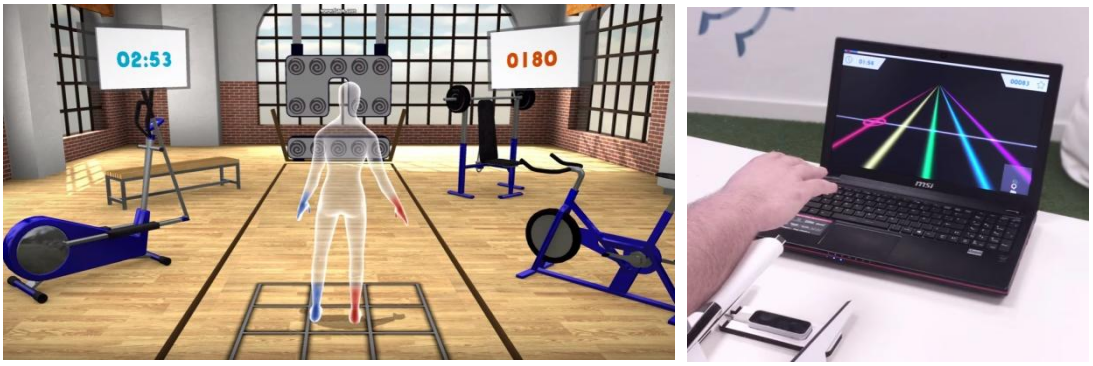


Figura 7: Juegos de VirtualRehab.

Un último ejemplo de rehabilitación con videojuegos es Visual Touch Therapy [19], un programa de software y plataforma de juegos que utiliza el controlador de gestos Leap Motion como un régimen de terapia asequible y que pueden utilizar en casa las personas con dificultades para usar sus habilidades motrices.

Incluye retos convencionales de los videojuegos que pueden ser vencidos usando ejercicios de movimientos repetitivos para proveer un método de terapia que, al

contrario que la ejercitación, es constantemente gratificante y da un feedback más inmediato que los métodos tradicionales.

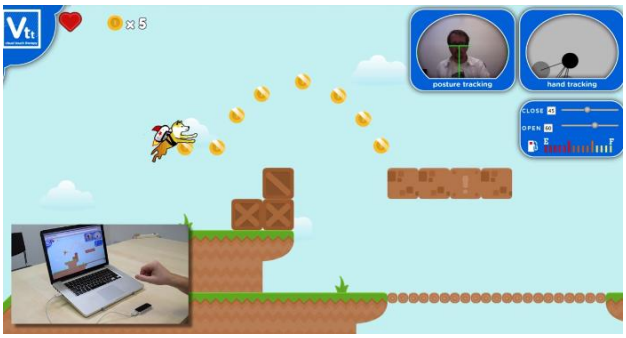


Figura 8: Juego Rocketdog de Visual Touch Therapy.

2.4 LEAP MOTION EN MEDICINA

La tecnología del Leap Motion está transformando el mundo de la atención médica con desarrollos en las aplicaciones por todo un gran rango de campos, desde terapia física, accesibilidad de necesidades especiales, investigación hospitalaria para entrenamiento quirúrgico e interfaces médicas. Con recientes innovaciones en módulos integrados, Leap Motion está ya haciendo posible a los cirujanos el acceso a información médica vital manteniendo un ambiente esterilizado [20].

De la sala de operaciones a la realidad virtual, hay 5 maneras en las que la gente está utilizando la tecnología del Leap Motion para aplicaciones médicas y de asistencia [21]:

1. Escaneo médico. Los rayos X y las imágenes de las resonancias magnéticas juegan un papel importante en las salas de operaciones en todo el mundo.

   Problema: La esterilización es esencial en las salas de operaciones. Los cirujanos no pueden tocar el ratón de un ordenador sin arriesgarse a contaminación cruzada.

   Solución: TedCas [22] está desarrollando una consola "enchufa y juega" que permite a los cirujanos navegar por las imágenes de forma rápida, intuitiva y segura. El dispositivo está en ensayos clínicos.

2. Pérdida de oído. Cientos de miles de personas usan el lenguaje de signos en su vida diaria.

Problema: Desde su casa al lugar de trabajo y durante todo el trayecto entre medias, las personas sordas se enfrentan a serias barreras de comunicación.

Solución: Una de las mejores 25 invenciones de 2014 publicadas en la revista TIME: MotionSavvy's UNI [23]. Es una herramienta de comunicación bidireccional para sordos que utiliza Leap Motion y tecnologías de voz.

Figura 9: UNI utilizando Leap Motion.

3. Temblores de manos. El Parkinson, la enfermedad de Wilson, la distonía y otras enfermedades dificultan la vida diaria de millones de personas.

   Problema: No hay muchas maneras en que los pacientes y doctores puedan rastrear la progresión en el tiempo de los temblores de forma rápida y fiable.

   Solución: Desarrolladores e investigadores han estado experimentando utilizando la tecnología del Leap Motion para medir temblores de mano.

4. Distorsiones en la vista. El estrabismo y los ojos vagos aparecen cuando el cerebro humano ignora la entrada de información por el ojo no dominante.

   Problema: No hay muchas maneras de entrenar el cerebro de estas personas más allá de los aburridos ejercicios clínicos.

   Solución: Vivid Vision [24] apuesta por la realidad virtual y los controles de movimientos para engañar al cerebro del jugador y fortalecer el ojo más débil. Su tecnología está ahora extendiéndose a clínicas oftalmológicas.

5. Terapia física. Después de un ictus u otra enfermedad paralizante, los pacientes tienen un largo recorrido para recuperar el funcionamiento normal de sus manos.

   Problema: Los ejercicios de rehabilitación pueden parecer poco productivos y repetitivos. Los pacientes a menudo necesitan permanecer en grandes instalaciones porque las tecnologías que se necesitan son caras y difíciles de instalar.

   Solución: El instituto Burke Medical Research [25], Ten Ton Raygun [26] y otros han diseñado juegos experimentales usando la tecnología del Leap Motion para pacientes que han sufrido derrames cerebrales.

Capítulo 3.

MEDIOS EMPLEADOS

En este capítulo se listan los medios empleados para la realización del proyecto. Para la elección del software y del hardware apropiado para realizar el *serious game* del proyecto se han realizado comparativas entre varios medios que se expondrán en el apartado correspondiente.

3.1. SOFTWARE

3.1.1. ESTUDIO DE MOTORES DE VIDEOJUEGOS

A la hora de crear un videojuego es fundamental elegir un adecuado motor de videojuegos que nos permita su diseño, creación y representación. Un motor de videojuegos regular aporta: scripting, renderizado de imágenes, inteligencia artificial, física, animación, cinemática, acceso a la red y gestión de recursos [2].

- Scripting: Permite a los desarrolladores escribir pequeños trozos de código para controlar ciertas partes del juego.
- Renderizado de imágenes: Es el corazón de la parte visual del juego. Maneja las luces, sombras, trazado de los rayos y renderizado de objetos 3D.
- Inteligencia artificial: Da vida al mundo y a los personajes del juego a través de una serie de rutinas que hace posible la interacción con el ambiente del juego.
- Animación: Añade comportamiento a los objetos a través de transformaciones, esqueletos, deformaciones y dinámicas.
- Física: Aporta interacción realista entre objetos y con el ambiente.
- Cinemática: Añade la posibilidad de incluir videos en el juego para llamar la atención del jugador.

- Acceso a la red: Aporta soporte para implementar el juego en un entorno de red, sea cliente-servidor o peer to peer.
- Gestión de recursos: Un asunto fundamental para los motores de videojuegos es el uso eficiente de los recursos del ordenador (CPU, tarjeta gráfica, memoria, almacenamiento, hardware) y la carga de los recursos del juego (animaciones, texturas, objetos 3D, sonidos, etc.).

Los motores de videojuegos pueden ser clasificados de acuerdo a diversos criterios, siendo uno de ellos el tipo de licencia: comercial y gratuito. Con este proyecto se busca crear un videojuego asequible para todos y, por tanto, se ha utilizado uno de licencia gratuita y un motor de código abierto (open source) de manera que pueda ser modificado en cualquier momento por entendidos de la programación con indicaciones de terapeutas para poder mejorar las características del videojuego y hacerlo más acorde a cada paciente.

A la hora de elegir un motor de videojuegos se han comparado principalmente Unreal Engine (UE), CryEngine y Unity, siendo estos los líderes del mercado dentro de los que cumplían los requisitos necesarios para este proyecto. A pesar de que tanto Unreal como CryEngine han sido utilizados para crear importantes videojuegos en la historia, la que es sin duda la plataforma de desarrollo de videojuegos que domina globalmente es Unity [27], la cual cuenta con el 45% de participación en el mercado de los motores de videojuegos como podemos ver en la figura 10.

Figura 10: Participación en el mercado global de los motores de videojuegos.

Las atracciones clave de por qué Unity es tan poderoso son la baja curva de aprendizaje para principiantes, el rápido desarrollo y la integración multiplataforma

(Unreal admite 10 plataformas y CryEngine 5 en comparación con las 25 de Unity). Es por esto que Unity cuenta con 5,5 millones de usuarios registrados [27]. A pesar de que esta curva de aprendizaje es muy positiva y que los principiantes lo elegirán frente a otros motores, existen casos en que los desarrolladores se ven limitados y en ese momento comienzan a utilizar Unreal Engine.

Unity juega un papel importante en el auge del mercado global de los videojuegos: más juegos son creados con Unity que con cualquier otra tecnología de videojuegos, más jugadores juegan con juegos hechos con Unity y más desarrolladores confían en sus herramientas y servicios para llevar su negocio. Son muchos los desarrolladores que usan Unity por su excelente funcionalidad, contenido de alta calidad y habilidad para ser usado en casi cualquier tipo de juego. Además, Unity lleva la delantera en el creciente mercado de la realidad virtual.

En relación a los otros dos motores (UE y CryEngine) destacar que ambos superan a Unity en la calidad de gráficos y realismo, pero puesto que los gráficos no son la característica más relevante de este proyecto, el motor de videojuegos seleccionado ha sido Unity por todo lo anteriormente comentado, por su éxito, la facilidad de manejo, la gran cantidad de tutoriales y recursos y la compatibilidad e integración de realidad virtual con el Leap Motion. En la propia página de Leap Motion se puede encontrar documentación para la importación y uso de los recursos y scripts del Leap Motion en Unity.

En el siguiente apartado se amplía la información acerca del motor de videojuegos escogido: Unity.

3.1.2. UNITY

Figura 11: Logotipo de Unity.

Unity es un motor de videojuegos creado por Unity Technologies y se usa para desarrollar videojuegos para ordenador, móviles y páginas web. Está disponible como plataforma de desarrollo para Windows, OSX y Linux y es compatible con las plataformas que aparecen en la figura 12.

Figura 12: Plataformas compatibles con Unity [28].

El éxito de Unity ha llegado en parte debido al enfoque en las necesidades de los desarrolladores independientes que no pueden crear ni su propio motor del juego ni las herramientas necesarias o adquirir licencias para utilizar plenamente las opciones que aparecen disponibles. El enfoque de la compañía es "democratizar el desarrollo de juegos" y hacer el desarrollo de contenidos interactivos en 2D y 3D lo más accesible posible a tantas personas en todo el mundo como sea posible [29]. Con el objetivo de adaptarse al uso que le quiere dar cada usuario, existen varias licencias para desarrolladores (figura 13) y todas ellas ofrecen documentación y tutoriales.

Figura 13: Planes disponibles de Unity.

Para programar el código del videojuego en Unity se pueden utilizar tanto MonoDevelop como Microsoft Visual Studio utilizando los siguientes lenguajes de programación:

- C#, utilizado en un gran porcentaje de los scripts creados.
- UnityScript, o JavaScript para Unity.
- Boo, con una sintaxis inspirada en Python.

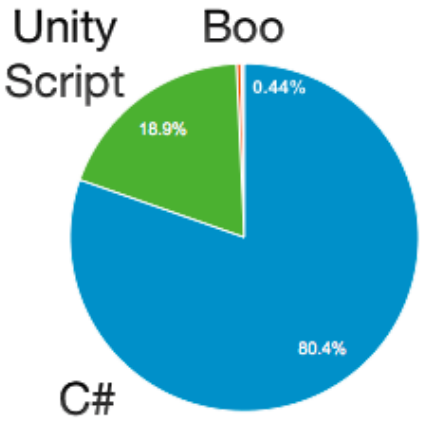


Figura 14: Uso de los lenguajes de programación en Unity [30].

Como se puede observar en la figura 14, muy poca gente utiliza Boo, es por ello que Unity decidió abandonar el soporte a la documentación de Boo cuando se lanzó Unity 5.0 y así usar los recursos de una manera más constructiva.

Para facilitar el desarrollo de videojuegos a los usuarios, en 2010 se puso en funcionamiento el Unity Asset Store, donde se pueden comprar recursos de distintas categorías. Actualmente incluye más de 15.000 (gratis y de pago) [31] modelos 3D, extensiones del editor, scripts, texturas, materiales, audios y animaciones que pueden ser fácilmente añadidos al juego.

Para dar un uso correcto a los recursos en el videojuego, Unity pone a disposición de los usuarios la tabla 4 que indica los formatos compatibles con el motor. Además, Unity puede usarse junto con 3ds Max, Maya, Softimage, Blender, Modo, ZBrush, Cinema 4D, Cheetah3D, Adobe Photoshop, Adobe Fireworks y Allegorithmic Substance. Los cambios realizados a los objetos creados con estos productos se

actualizan automáticamente en todas las instancias de ese objeto durante todo el proyecto sin necesidad de volver a importar manualmente.

FORMATOS DE IMAGEN		
.psd	.jpg	.png
.gif	.bmp	.tga
.tiff	.iff	.pict
.dds		

FORMATOS DE AUDIO		
.mp3	.ogg	.aiff
.wav	.mod	.it
.sm3		

FORMATOS DE VIDEO		
.mov	.avi	.asf
.mpg	.mpeg	.mp4

FORMATOS DE TEXTO		
.txt	.htm	.html
.xml	.bytes	

Tabla 4: Formatos compatibles con Unity.

Adicionalmente, en Unity se muestra todo el juego y las variables mientras el desarrollador juega, además estas pueden ser alteradas sobre la marcha sin necesidad de escribir ni una sola línea de código. El juego puede ser pausado en cualquier momento o pasar el código de una en una instrucción.

3.1.3. MICROSOFT VISUAL STUDIO

Se trata de un entorno de desarrollo integrado para sistemas operativos Windows. Soporta múltiples lenguajes de programación como C++, C#, Visual Basic .NET, F#, Java, Python, Ruby y PHP.

En este proyecto se ha utilizado Visual Studio 2015 para programar, en C# en este caso, los scripts necesarios para el desarrollo del *serious game* en Unity.

Figura 15: Logotipo de Visual Studio

3.2. HARDWARE

3.2.1. DISPOSITIVOS DE CAPTURA DE MOVIMIENTO

Desde un principio, para la elección del dispositivo de captura de movimiento para el proyecto, se valoraron la Kinect v2 y el Leap Motion. Aunque la Kinect también ha sido utilizada para la monitorización de movimientos precisos como por ejemplo los movimientos de los dedos, cuando el enfoque de la rehabilitación es la individualización de los dedos, el sensor del Leap Motion es mejor opción. El pequeño área de observación y la alta resolución del dispositivo diferencia el producto de la Kinect, que es más apropiada para el rastreo del cuerpo entero en un espacio del tamaño de un salón.

Además, entre los dispositivos cinemáticos de manos se han realizado estudios donde se evalúa el potencial de su uso en un marco clínico para rehabilitación de manos y el Leap Motion, por su bajo coste, precisión y facilidad de uso, resulta ser el más adecuado para la creación del *serious game* de este proyecto.

En la tabla 5 podemos encontrar la comparativa de los siguientes tres dispositivos:

- Leap Motion (utiliza cámara IR)
- Guante de datos 5-DT 14 (Sensor Data Glove)
- Guante de datos Vhand DGTech 5 (Sensor Data Glove)

Figura 16: Dispositivos cinemáticos de manos.

Los resultados demostraron que el Leap Motion es el más preciso y el más fácil de usar para sujetos con movilidad pero fallaba en recopilar la cinemática de las manos de pacientes que habían sufrido un derrame cerebral debido a la obstaculización de la vista por sus manos parcialmente cerradas.

Dispositivo	Leap Motion	Guante 5DT	Guante Vhand DGTech
Tecnología	Cámaras IR dual	Fibra óptica	Piezorresistivo
Precio	$80	$5420 cada guante	$800 por guante
Número de articulaciones	>14	14	5
Precisión del sensor	32 bit	8 bit	~4 bit
Frecuencia de muestreo	hasta 115Hz	200Hz	100Hz
Ventajas	· No requiere de calibración ni instalación · Traslación lineal del ángulo de articulación · Extremadamente barato · Está siendo desarrollado y mejorado activamente	· Funciona con cualquier posición y orientación de la mano · Alta resolución y frecuencia de muestreo para detectar el mínimo movimiento · Instalación rápida del guante	· Funciona con cualquier posición y orientación de la mano · El guante puede acomodarse a cualquier tamaño de mano
Desventajas	· Dificultad midiendo una mano cerrada · Requiere de una orientación específica de la mano para resultados óptimos · Limitado a sujetos con movilidad · Puede perder el rastro de la mano causando saltos erróneos en las medidas · Dificultad midiendo manos pequeñas · Utiliza un alto % de la tarjeta USB	· Precio elevado · Método de calibración prolongado para la medición de los ángulos de las articulaciones · Calibración no lineal hace que las medidas próximas a los límites del sensor sean altamente imprecisas · Requiere guantes adicionales para manos pequeñas	· Baja resolución del muestreo no puede capturar movimientos pequeños · Calibración no lineal hace que las mediciones próximas a los límites del sensor sean altamente imprecisas · Instalación prolongada · Sensores muy variables de baja calidad · Difícil de inicializar con el ordenador
Aplicaciones	· Sujetos no distónicos con casi pleno alcance de la mano · Desarrollo de software	· Sujetos con rango de movimiento de mano limitado	· Sujetos con manos pequeñas · Sujetos que tengan dificultades para ponerse guantes

Tabla 5: Comparativa de dispositivos para captura de movimientos de las manos [32].

Para la mayor parte de usos, incluyendo el desarrollo de las aplicaciones de rehabilitación, el Leap Motion es el más barato, el más rápido de instalar y aporta los resultados más claros.

En el siguiente apartado se amplía la información acerca del dispositivo escogido para el proyecto: el Leap Motion.

3.2.2. LEAP MOTION

Figura 17: Logotipo de Leap Motion.

El Leap Motion es un pequeño dispositivo USB (80x30x11mm) diseñado para colocar en ordenadores o cascos de realidad virtual y que se lanzó por primera vez en 2013. Se trata de un sensor que nos permite controlar el ordenador a base de gestos en el aire usando las manos y los dedos, ya que lo que hace es trazar una imagen virtual de nuestras manos y articulaciones y rastrear todos los movimientos.

Usa dos cámaras infrarrojas monocromáticas que generan casi 200 fotogramas por segundo y tres LEDs infrarrojos. El dispositivo observa el área hemisférica que podemos observar en la figura 18 y la información es enviada a través de un cable USB al ordenador, donde es analizado por el software del Leap Motion. En 2013 un estudio señaló que el promedio de precisión del controlador es de 0,7mm.

Figura 18: Área de interacción del Leap Motion.

En la figura 18 se define un área de interacción de aproximadamente entre 60 y 80cm por encima del dispositivo, por 60cm de ancho en cada lado (ángulo de 150º), por 60cm de profundidad en cada lado (ángulo de 120º) [33]. El rango de detección por encima del dispositivo se debe a que antiguamente era de 60cm pero actualmente, con el software Orion beta, se ha extendido hasta los 80cm. Pero este rango además está limitado por la propagación de la luz de los LEDs en el espacio, resultando más difícil intuir la posición de las manos en 3D más allá de cierta distancia, y la intensidad de estas luces está básicamente limitada por la corriente máxima que puede pasar por la conexión del USB.

A pesar de que el Leap Motion no ha sido específicamente desarrollado para rehabilitación (sino para la interacción humana con el ordenador especialmente relacionada con los juegos), se trata de un dispositivo adecuado para la rehabilitación con videojuegos debido a su bajo coste, pequeñas dimensiones en comparación con otros dispositivos y, especialmente, su fácil uso, ausencia de marcadores y los aspectos cautivadores de su tecnología. Asimismo, provee información a los desarrolladores sobre las manos y dedos como la posición de la punta de los dedos, velocidad y dirección de manos y dedos, longitud y ancho de los dedos, etc., información que no puede ser recogida bajo la observación humana.

Similar a otros dispositivos que se usan en terapia de videojuegos, el hecho de que el objetivo de su uso se haya extendido a la rehabilitación puede deberse al incremento de la participación de pacientes, tanto en términos del tiempo empleado en entrenar como en la implicación cognitiva del entrenamiento. Pero además, a diferencia de otros dispositivos, el Leap Motion puede incluir en sus terapias de videojuegos también a pacientes ancianos, en silla de ruedas (criterio de exclusión en estudios que utilizan, por ejemplo, la Kinect) y que tengan dificultades manejando controladores (criterio de exclusión para usar, por ejemplo, el controlador de la Nintendo Wii).

La simplicidad del Leap Motion puede facilitar el acercamiento de los pacientes a la tecnología, aumentando su sensación de inmersión en el ambiente virtual, su imaginación y su interacción con el ambiente virtual. Estos tres conceptos (inmersión, imaginación e interacción) se conocen como las tres I's de la realidad virtual.

Figura 19: Posicionamiento del Leap Motion.

Otro factor importante a valorar del dispositivo es su seguridad, ya que es un requisito vital en los dispositivos que se comercializan para rehabilitación que no son específicamente desarrollados para uso médico. El Leap Motion Controller es un sistema optoelectrónico donde los pacientes solo mueven las manos en el aire sin tener contacto con el dispositivo. Se rastrean ambas manos, los diez dedos e incluso herramientas (con formas parecidas a los dedos como, por ejemplo, un bolígrafo), modelando todas las articulaciones fisiológicas y debe ser situado entre el usuario y la pantalla del ordenador como se muestra en la figura 19.

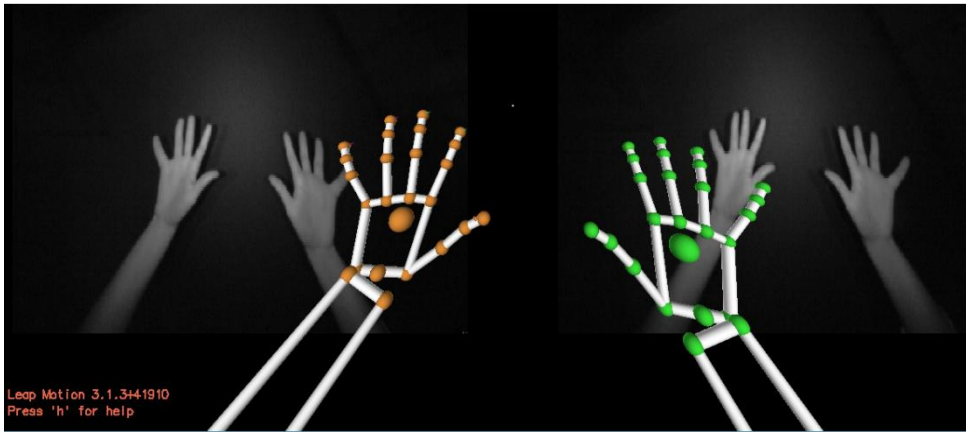


Figura 20: Vista en infrarrojo desde el visualizador del Leap Motion.

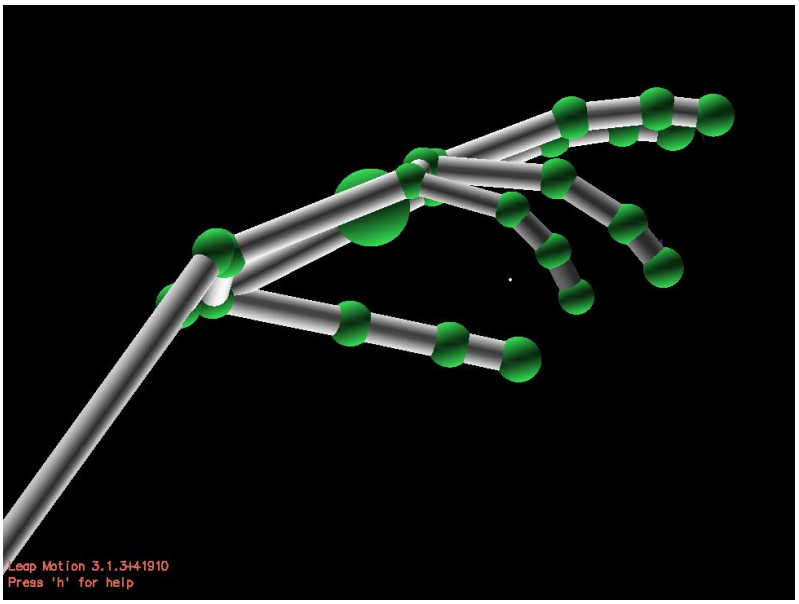


Figura 21: Vista lateral desde el visualizador del Leap Motion.

En términos de reconocimiento de gestos, hasta la fecha el SDK del Leap Motion aporta los siguientes cuatro gestos (sin necesidad de entrenamiento para su reconocimiento): *key tap* (gesto que imita un dedo pulsando una tecla), *screen tap* (gesto que imita un dedo presionando una pantalla vertical), *swipe* (barrido lineal de la mano) y *circle* (movimiento de un dedo trazando un círculo).

El SDK del Leap Motion está disponible para Windows, Linux y OSX y los lenguajes de programación que admite son C++, C#, Java, Javascript y Python. Para todos ellos se puede encontrar documentación en la página oficial de Leap Motion, además de documentación para las plataformas con las que el controlador se puede integrar en realidad virtual: Unity y Unreal. Para poder trabajar en realidad virtual es necesario descargarse el software Orion Beta.

Figura 22: Documentación en la web del Leap Motion.

Por último, en su tienda Leap Motion App Store podemos encontrar una gran variedad de aplicaciones, incluyendo aplicaciones neurológicas.

3.2.3. OCULUS RIFT

Gracias al software Orion Beta del Leap Motion, es posible jugar al *serious game* que se ha creado para este proyecto en realidad virtual con las Oculus Rift. Si bien es cierto que jugar al *serious game* sin las Oculus es muy asequible, para poder utilizar las gafas es necesario contar con un ordenador potente que cumpla con las características que estas requieren. Para ello en este proyecto se ha utilizado el ordenador que aparece en el siguiente apartado.

Figura 23: Leap Motion colocado en las Oculus Rift.

Para utilizar el Leap Motion con las Oculus, el dispositivo debe ser incorporado a ellas como se observa en la figura 23. Este detectará las manos en frente a él y no encima como se indicaba en el apartado del Leap Motion, pero esto no afectará a la manera en que el usuario visualice sus manos en el juego ya que Orion Beta está preparado para ambas opciones de juego.

3.2.4. ORDENADOR

Un ordenador potente, tanto para crear como para jugar a videojuegos en realidad virtual con las Oculus Rift, es un requisito fundamental. Para el desarrollo de este *serious game* se ha utilizado un ordenador cuyas características se adecuasen a las exigidas por las Oculus: procesador potente, buena tarjeta gráfica, alta capacidad de memoria... Finalmente, se ha utilizado el MSI GT72 2QE DOMINATOR PRO.

Figura 24: MSI GT72 2QE DOMINATOR PRO

PANTALLA	17,3 pulgadas LED FullHD, 1920x1080 píxeles
PROCESADOR	Intel Core i7-4710MQ 4 núcleos (8 hilos) a 2,5~3,5 GHz. 22 nm., TDP de 47 vatios.
GRÁFICA INTEGRADA	Intel HD Graphics 4600 400 MHz. ~ 1,15 GHz.
GRÁFICA DEDICADA	NVidia GeForce GTX 880M 954 MHz. + Boost, 1.536 núcleos CUDA 8 GB GDDR5 256bits
RAM	16 GB DDR3 (2x8 GB)
ALMACENAMIENTO	SSD 256 GB mSATA (2x128 GB en RAID 0) HDD 1 TB 7.200 rpm SATA3
ENERGÍA	Batería de 9 celdas Fuente de alimentación de 230 vatios
OTROS	Trackpad y teclado SteelSeries con retroiluminación Killer E2200 (Ethernet Gigabit) + Killer N1525 (WiFi ac) 2xDisplayPort + 1xVGA + 1xHDMI Bluetooth 4.0 Audio Sound Blaster Cinema 2 Dynaudio
DIMENSIONES	427,9x293,8x47,7 milímetros
PESO	3,8 kilogramos de peso

Figura 25: Especificaciones técnicas del MSI.

Capítulo 4.

DESARROLLO DEL SERIOUS GAME

En este apartado se explicarán las especificaciones del *serious game* realizado para este proyecto, se describirá su funcionamiento y el proceso de desarrollo que se ha realizado con el software y hardware descritos en el capítulo 3. Por último, se expondrán brevemente otros trabajos realizados durante el desarrollo de este proyecto.

4.1. ESPECIFICACIONES DEL SERIOUS GAME

Se trata de un videojuego para la rehabilitación de manos y dedos en el que el sujeto ejercitará la disociación de los dedos moviéndolos de forma independiente además de la coordinación bimanual requiriendo la involucración de ambas manos para lograr la meta. Asimismo, el juego incorpora la característica adicional de motivación y diversión que convertirán la terapia en un entretenimiento.

Una de las comunidades de pacientes que más puede beneficiarse de la interacción de las manos son los que han sufrido un derrame cerebral. El derrame cerebral causa hemiplejia que afecta a las funciones motrices de uno de los lados del cuerpo, típicamente, los pacientes pierden parte o completo control de una de sus manos [34]. Pero el público objetivo para este s*erious game* no está limitado sino que puede servir para cualquier tipo de paciente para el que los especialistas consideren de utilidad, ya sean personas con trastornos neurológicos incluyendo Parkinson, parálisis cerebral, esclerosis múltiple, ataxia, etc. o sujetos que hayan sufrido quemaduras o lesiones físicas.

Además, como se ha mencionado en el apartado del Leap Motion, se ha escogido este dispositivo con la finalidad de no tener que hacer exclusiones por limitaciones como pacientes en sillas de ruedas, con sordera, ancianos o pacientes con dificultades para

agarrar controladores, ya que para manejar el Leap Motion tan solo se necesita mover las manos en el aire e incluso se podría colocar un soporte en el brazo si así lo requiriese el paciente como podemos observar en la figura 26.

Figura 26: Soporte utilizado por VirtualRehab [18].

En definitiva, el *serious game* dotará de las siguientes características:

- Asequible para cualquier tipo de paciente y sus requisitos.
- Opción de adaptarse a distintas patologías y manos del paciente.
- Registro de los resultados para su evaluación.
- Fácil de entender y manejar.
- Añade una componente de entretenimiento, motivación e incluso competitividad entre usuarios que podría resultar beneficiosa.

4.2. DESCRIPCIÓN DEL SERIOUS GAME

Piano es un juego en el que se irán iluminando las teclas de un piano virtual y el paciente deberá pulsarlas con el dedo correcto ya que cada tecla se corresponde con un dedo de la mano.

En la primera pantalla (figura 27) se solicitarán los datos más relevantes del paciente que se deberán introducir por teclado. A partir de ellos se creará automáticamente un

fichero donde se irá guardando dicha información junto a los resultados de los ejercicios para la futura evaluación de un profesional. En el caso de que dicho usuario ya haya jugado en anteriores ocasiones, el juego reconocerá su nombre y apellido e incluirá los resultados de esta sesión en el mismo archivo que las anteriores para facilitar la evaluación al terapeuta y analizar los progresos del paciente.

Figura 27: Pantalla inicial del juego.

Al presionar el botón de Play, aparecerá la pantalla donde se permitirá el ajuste de las teclas a través de unos deslizadores (figura 28):

- Distancia entre teclas. Se personalizará dicha distancia a la disociación de los dedos que requiera la terapia.
- Grosor de las teclas. Se adaptará dicho grosor para que el paciente se sienta cómodo al pulsar las teclas.
- Altura de las manos. El paciente colocará las manos encima del dispositivo Leap Motion a la altura donde se sienta más cómodo y con el deslizador correspondiente se colocarán las manos virtuales de modo que queden apoyadas encima del teclado.

Figura 28: Pantalla de ajuste del teclado.

Una vez adaptado el formato al paciente, se pulsará el botón de siguiente y comenzará el juego. Es conveniente, por tanto, antes de cambiar de escena animar al usuario a que practique pulsando las teclas para que se familiarice con el entorno virtual.

El primer ejercicio se realizará con tan solo la mano derecha (figura 29 y 30). Las teclas irán cambiando a color negro en orden, indicando así cuál debe ser presionada y, a su vez, se iluminará la punta del dedo con el que se deba pulsar, correspondiéndose de la siguiente manera:

- La primera tecla (a la izquierda del todo): dedo pulgar de la mano derecha.
- Segunda tecla: dedo índice de la mano derecha.
- Tercera tecla: dedo corazón de la mano derecha.
- Cuarta tecla: dedo anular de la mano derecha.
- Quinta tecla (a la derecha del todo): dedo meñique de la mano derecha.

Cada tecla tan solo reaccionará al ser pulsada por el dedo que le corresponde, ignorando cualquier interacción con el resto de dedos con la finalidad de que se puedan evaluar los resultados correctamente, ya que, el tiempo que se tarde en pulsar cada tecla será cronometrado. Asimismo, tan solo sonará la tecla que corresponda tocar.

Figura 29: Pantalla previa al ejercicio con la mano derecha.

Figura 30: Usuario realizando el ejercicio con la mano derecha.

Una vez que se presione la tecla en negro, se reproducirá el sonido y pasará a la siguiente y así sucesivamente, guardando todos estos tiempos en el archivo junto al resto de resultados (figura 31).

Se realizarán dos series en que las teclas se colorearán de negro en orden de izquierda a derecha (es decir, se irá pulsando de pulgar a meñique) y otras dos series en que se

colorearán aleatoriamente. A la hora de guardar los datos se harán corresponder los tiempos (en segundos) con su respectivo dedo en el siguiente orden:

- Primer dato (T1): tiempo empleado en pulsar la tecla en la primera serie que se realiza en orden.
- Segundo dato (T2): tiempo empleado en pulsar la tecla en la segunda serie que se realiza en orden.
- Tercer dato (T3): tiempo empleado en pulsar la tecla en la primera serie que se realiza aleatoriamente.
- Cuarto dato (T4): tiempo empleado en pulsar la tecla en la segunda serie que se realiza aleatoriamente.

NOMBRE: Cynthia Cubeiro				
FECHA: 24/10/2016 15:33:42				
MANO LESIONADA: Izquierda				
MOTIVO DE LA REHABILITACION: Prueba				
T1 es el tiempo empleado en pulsar durante la primera serie en orden				
T2 es el tiempo empleado en pulsar durante la segunda serie en orden				
T3 es el tiempo empleado en pulsar durante la primera serie aleatoria				
T4 es el tiempo empleado en pulsar durante la segunda serie aleatoria	T1 (s)	T2 (s)	T3 (s)	T4 (s)
TIEMPOS DURANTE LA PRUEBA CON LA MANO DERECHA				
Pulgar derecho:	1.759	1.283	1.315	1.165
Indice derecho:	1.332	0.949	1.134	0.801
Corazon derecho:	1.116	1.399	1.015	0.999
Anular derecho:	1.333	2.217	1.167	0.699
Meñique derecho:	1.366	1.617	2.4	2.184
TIEMPOS DURANTE LA PRUEBA CON LA MANO IZQUIERDA				
Pulgar izquierdo:	2.465	3.014	2.283	1.165
Indice izquierdo:	2.05	2.3	2.582	0.801
Corazon izquierdo:	1.914	2.564	3.217	0.017
Anular izquierdo:	3.467	2.953	2.715	2.733
Meñique izquierdo:	2.719	2.367	0.4	2.199
TIEMPOS DE LA MANO IZQUIERDA DURANTE LA PRUEBA CON AMBAS MANOS				
Pulgar izquierdo:	3.134	2.201	2.699	1.033
Indice izquierdo:	2.116	1.866	1.731	1.148
Corazon izquierdo:	2.749	3.05	2.399	4.267
Anular izquierdo:	1.534	2.149	2.198	4.65
Meñique izquierdo:	2.483	2.048	1.183	2
TIEMPOS DE LA MANO DERECHA DURANTE LA PRUEBA CON AMBAS MANOS				
Pulgar derecho:	2.05	1.5	1.383	1.033
Indice derecho:	1.133	0.749	1.351	1.148
Corazon derecho:	0.815	1.499	2.233	1.167
Anular derecho:	0.033	0.616	1.366	1.516
Meñique derecho:	0.25	1.235	2.183	0.916

Figura 31: Archivo Excel con los datos y resultados del paciente.

Con estos tiempos en el documento Excel, el terapeuta podrá sacar promedios, gráficas y lo que resulte necesario para evaluar al paciente.

El siguiente ejercicio será el mismo que el anterior, con dos series de pulsaciones en orden y dos aleatorias, pero esta vez deberá realizarse con la mano izquierda. En este caso las teclas se corresponden de la siguiente manera:

- La primera tecla (a la izquierda del todo): dedo meñique de la mano izquierda.
- Segunda tecla: dedo anular de la mano izquierda.
- Tercera tecla: dedo corazón de la mano izquierda.
- Cuarta tecla: dedo índice de la mano izquierda.
- Quinta tecla (a la derecha del todo): dedo pulgar de la mano izquierda.

Figura 32: Pantalla previa al ejercicio con la mano izquierda.

Figura 33: Usuario realizando el ejercicio con la mano izquierda.

Finalmente se realizará el ejercicio con las dos manos para fomentar la coordinación bimanual. De nuevo se realizarán dos series en orden y dos aleatorias, esta vez involucrando todos los dedos de ambas manos. Las cinco teclas de la izquierda corresponden a la mano izquierda y las teclas de la derecha a la mano derecha y la serie comenzará por el meñique de la mano izquierda y finalizará con el meñique de la mano derecha.

Figura 34: Pantalla previa al ejercicio con ambas manos.

Figura 35: Usuario realizando el ejercicio con ambas manos.

Al finalizar todos los ejercicios podremos acceder, presionando el icono de la copa abajo a la derecha de la figura 35, a la pantalla de los resultados. Es en este momento cuando el paciente podrá comparar los tiempos que ha tardado en pulsar las teclas con cada dedo de cada mano, siendo la altura de las barras proporcional al promedio del tiempo empleado, e incitar su motivación para mejorar los resultados en futuras sesiones.

Figura 36: Pantalla con los resultados de los ejercicios.

En la figura 36 se observa como el usuario ha sido más lento pulsando las teclas con la mano izquierda que, como se recogía en la figura 31, se trataba de la mano afectada. Cada barra de la imagen pertenece al promedio de los tiempos de cada dedo, correspondiéndose un color con un tipo de dedo como se muestra en la leyenda de la pantalla.

Es importante mencionar que en cualquier momento y durante cualquiera de los ejercicios que se esté realizando se pueden efectuar las siguientes acciones:

- El icono de salir que se encuentra arriba a la izquierda de la pantalla finalizará el juego. La pantalla que aparece en la figura 37 aparecerá por si se hubiese pulsado el icono de salir por error.
- El icono de configuración, arriba a la derecha, llevará a la pantalla en que se ajustaba el teclado y una vez se finalice la adaptación regresará a la misma pantalla desde donde se realizó la petición.
- Las flechas hacia la izquierda y derecha permiten pasar de un ejercicio a otro, guardándose los tiempos que se hayan realizado, en el caso de que los haya, y permitiendo darse el caso de no realizar alguno de los ejercicios o hacerlos en distinto orden al establecido.

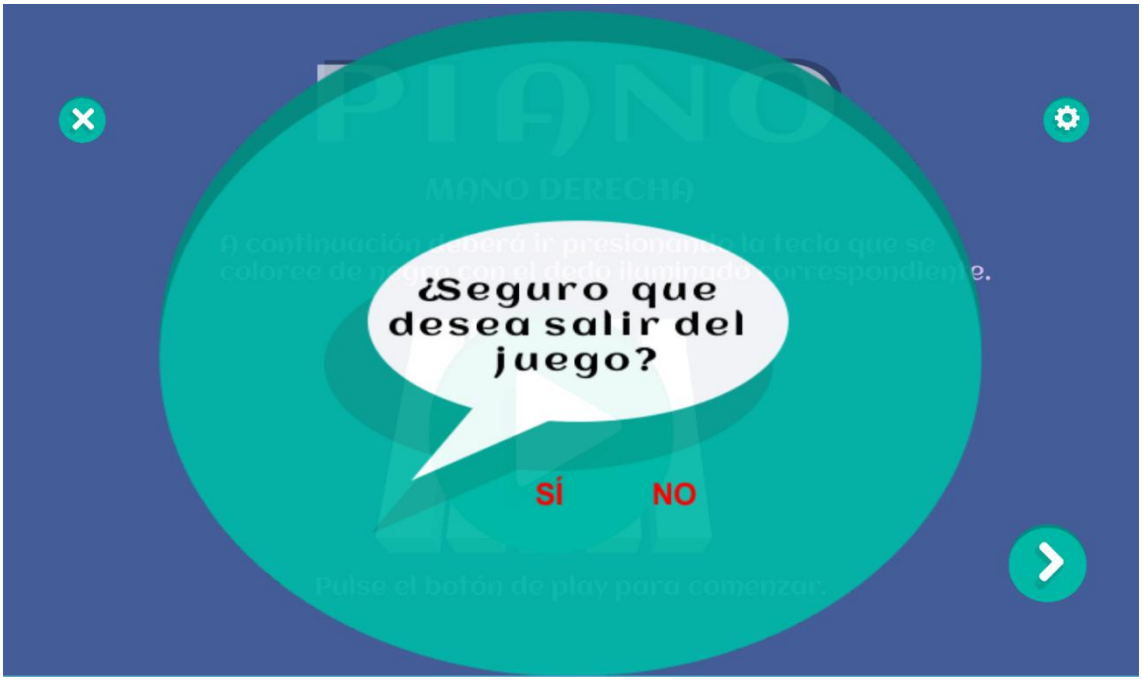


Figura 37: Pantalla de abandono del juego.

En definitiva, el proyecto está dividido en las siguientes seis escenas:

- Scene0_Intro: Pantalla inicial donde se recogen los datos del usuario.
- Scene1_Ajuste: Pantalla donde se adapta el teclado.
- Scene2_ManoDcha: Juego con la mano derecha.
- Scene3_ManoIzq: Juego con la mano izquierda.
- Scene4_AmbasManos: Juego con ambas manos.
- Scene5_Resultados: Pantalla donde se imprime un diagrama de barras con el promedio de los tiempos de los ejercicios.

Figura 38: Listado de escenas en Unity.

4.3. DESARROLLO DEL JUEGO "PIANO"

Para entender bien como se ha desarrollado el *serious game*, lo primero de este apartado será explicar cómo funciona Unity y, tras ello, la integración del Leap Motion en él. Se añaden en el tercer apartado los diagramas de flujo y en el último se describirán los scripts que se han programado para el correcto funcionamiento del juego.

4.4.1. CÓMO FUNCIONA UNITY

Como podemos observar en la figura 39, la pantalla de Unity se divide en las siguientes pestañas:

- Scene View (vista de la escena). En ella podremos ver los distintos objetos que se han incorporado a la escena y, directamente sobre ella, se pueden variar las posiciones, orientaciones, tamaños, etc.
- Game View (vista de juego). El desarrollador puede visualizar en esta pantalla cómo se verá la escena cuando juegue el usuario. Cuando se pulsa el icono de play que se encuentra arriba, se puede incluso simular el juego, pudiendo hacer mientras variaciones en la vista de la escena que se verán inmediatamente reflejadas en el juego.
- Hierarchy (jerarquía). En esta pestaña aparecerán todos los elementos de la escena: cámaras, objetos, scripts, botones, textos, etc.
- Project (proyecto). Aquí aparecerán todos los recursos que podamos necesitar para el proyecto: imágenes, audios, materiales, texturas, escenas, scripts, prefabricados, etc. En la figura 39, por ejemplo, podemos ver la lista de escenas que se han utilizado para el *serious game*, siendo la lista de elementos que aparecen en la jerarquía los pertenecientes a la escena en que se adapta el teclado al usuario (Level1_Ajuste).
- Inspector. En esta pestaña aparecen todas las propiedades del elemento seleccionado en la jerarquía. Contiene su posición, tamaño, materiales, texturas... y se incluyen, scripts, audios, etc.

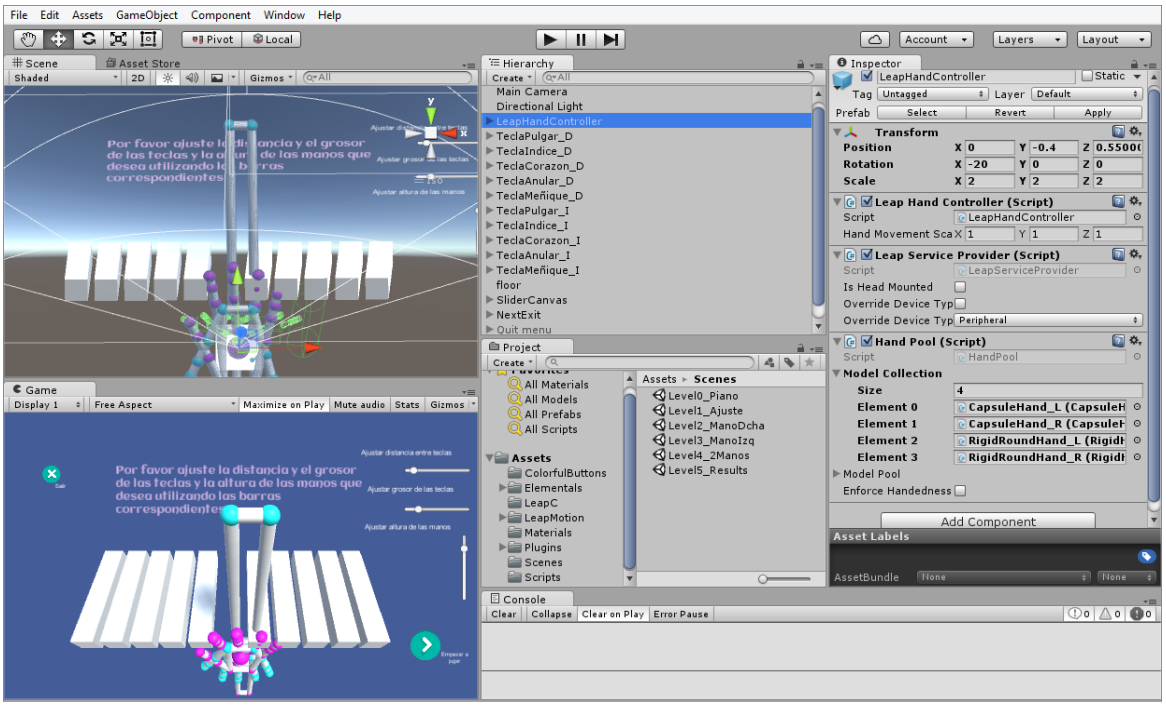


Figura 39: Pantalla de trabajo de Unity.

4.4.2. LEAP MOTION EN UNITY

Para empezar, hay que añadir el controlador del Leap Motion a la escena y así poder interactuar con él durante el juego. Para ello, siguiendo las instrucciones que se encuentran en la documentación en la página oficial de Leap Motion [35], hay que descargarse el paquete de recursos e importarlo desde Unity.

En la pestaña *Project*, dentro de los recursos, aparecerán tres nuevas carpetas relativas al Leap Motion como podemos observar en la figura 40 (LeapC, LeapMotion y Plugins). Dentro de la carpeta LeapMotion encontraremos el controlador y los prefabricados que representarán las manos del usuario durante el juego. Estas deben ser añadidas a la jerarquía de la escena como, de nuevo, podemos observar en la figura 40, de modo que durante el juego aparecerán las manos que se muestran en la *game view* de la imagen.

Este diseño de las manos viene perfectamente desglosado con sus dedos y estos a su vez fraccionados en sus tres falanges. Esto permite sacar el máximo partido de ellas,

por ejemplo, a la hora de sacar datos o agregarle una funcionalidad, como en este proyecto, añadirle luz a la falange distal para señalizar con qué dedo se debe pulsar la tecla.

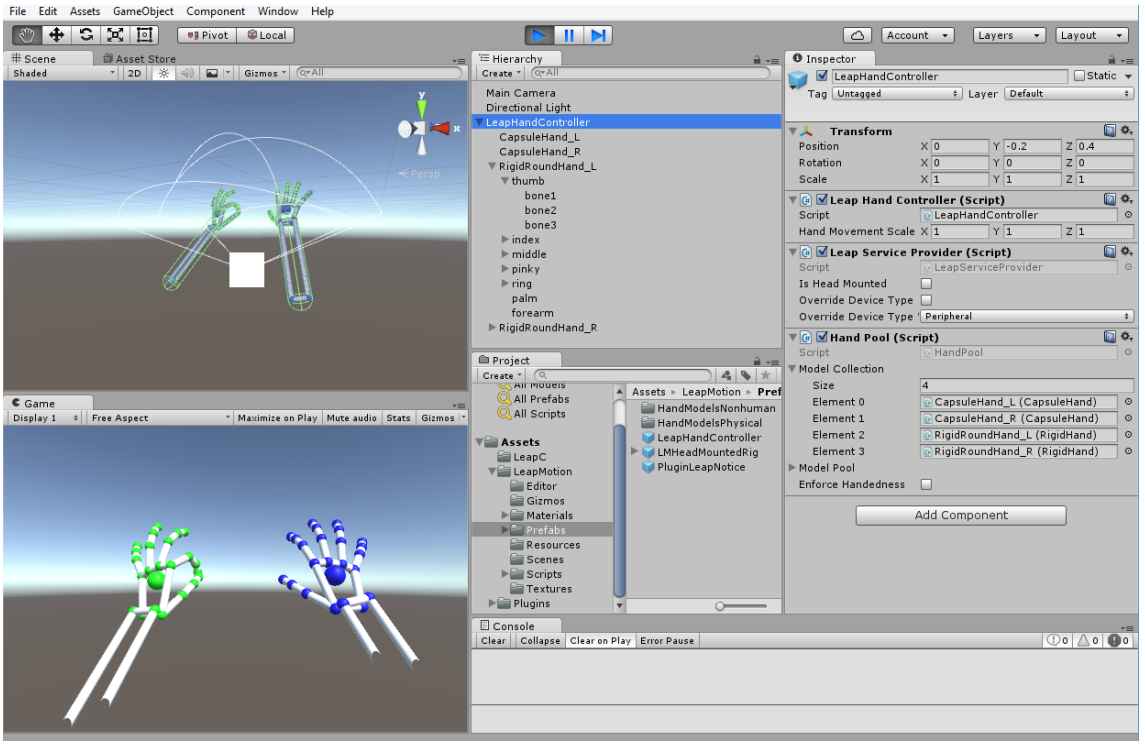
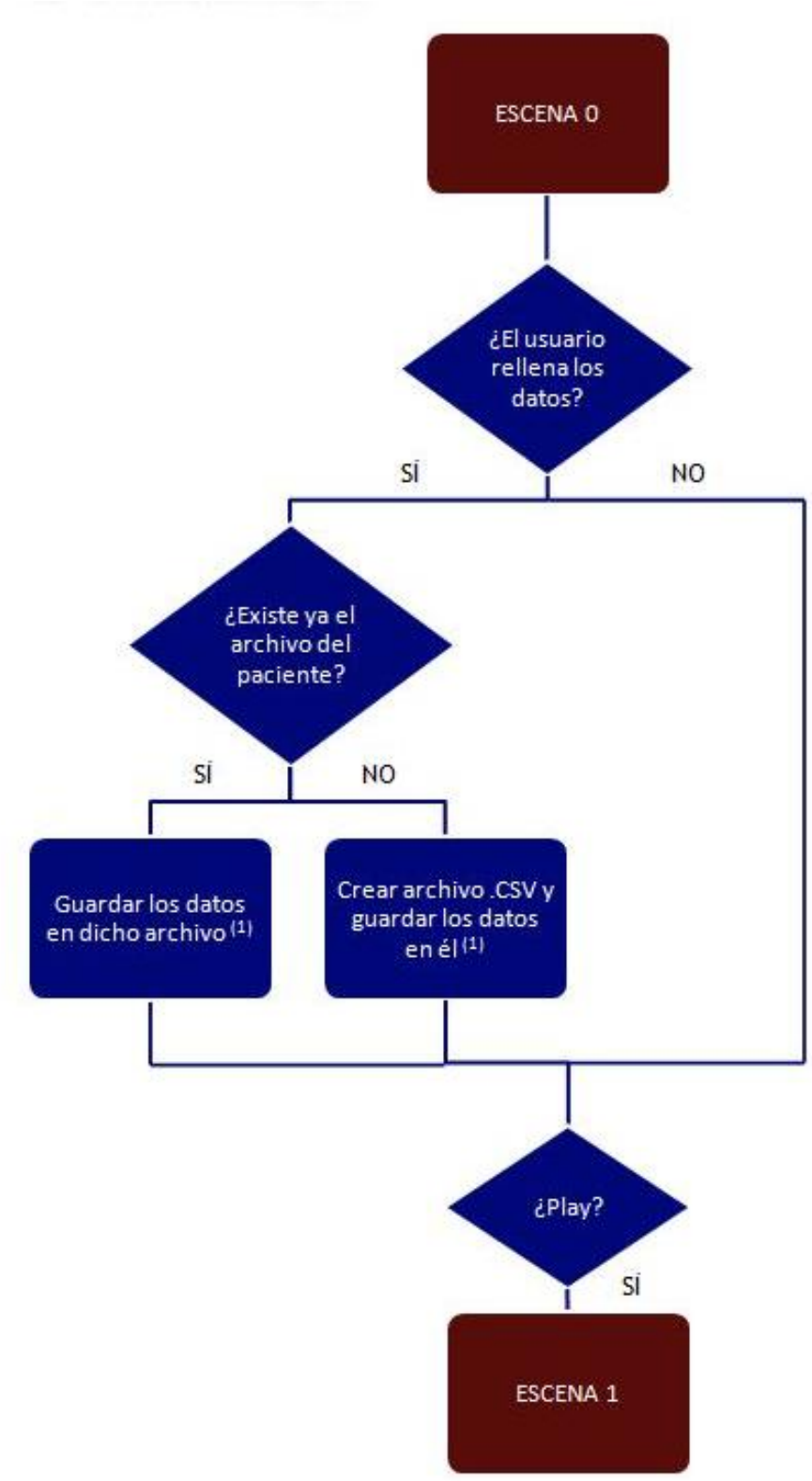


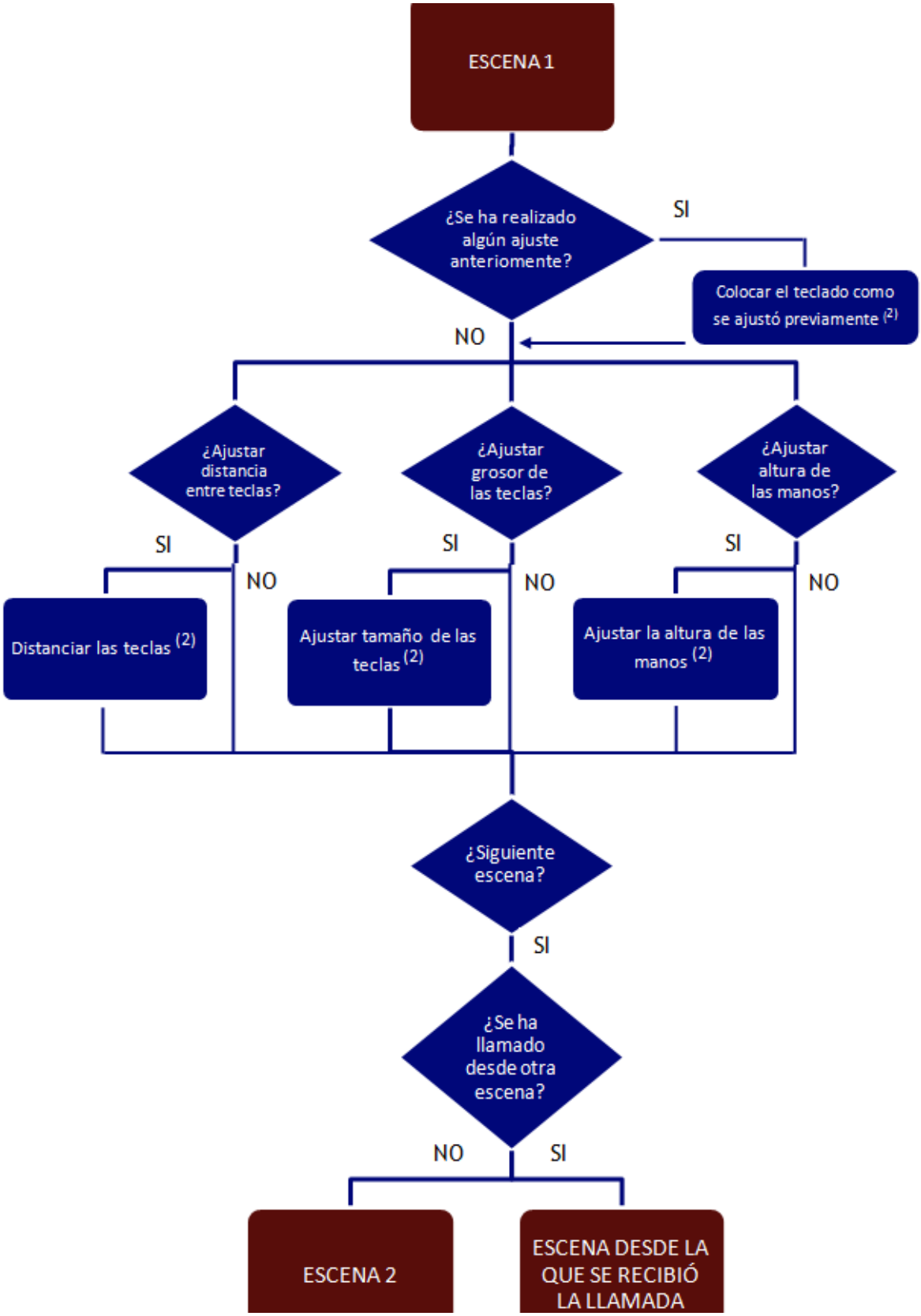
Figura 40: Prefabricado de las manos del Leap Motion en Unity.

4.4.3. DIAGRAMAS DE FLUJO

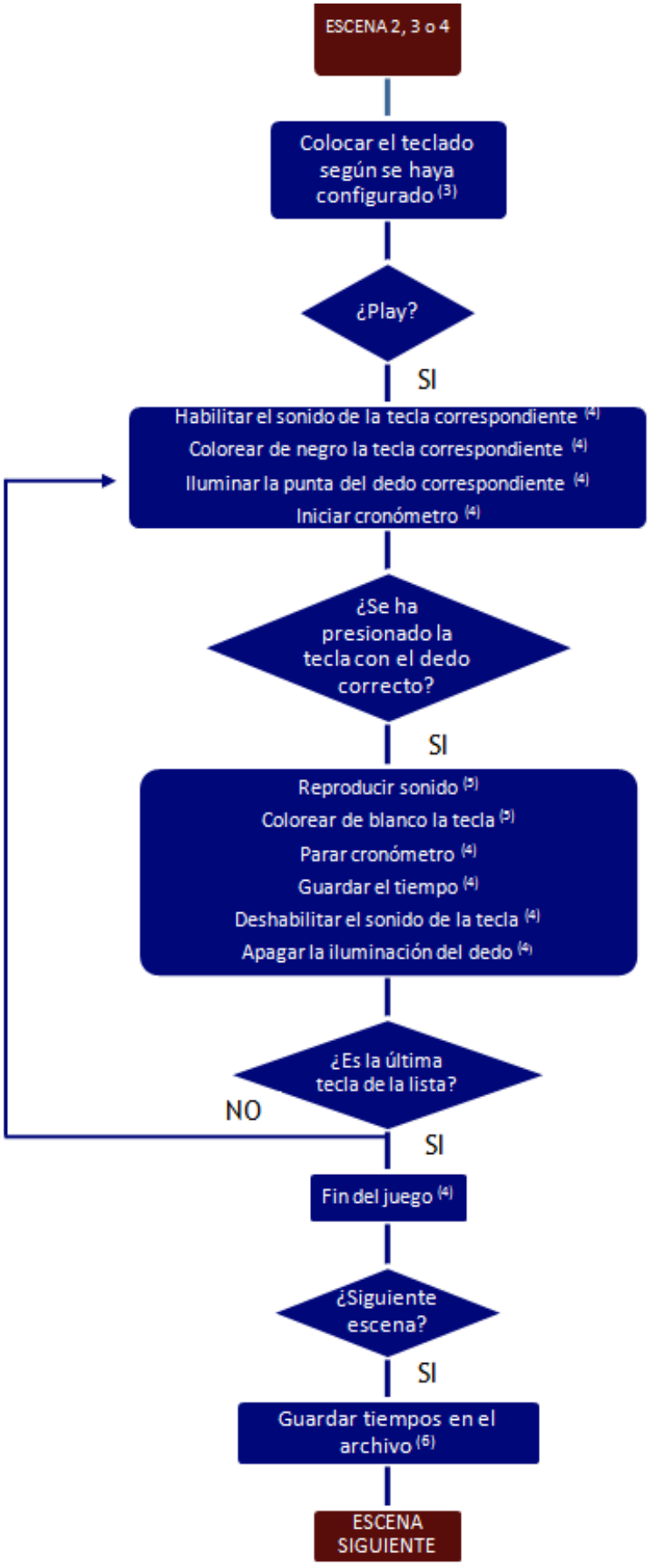
ESCENA 0: Pantalla inicial



ESCENA 1: Pantalla de ajustes

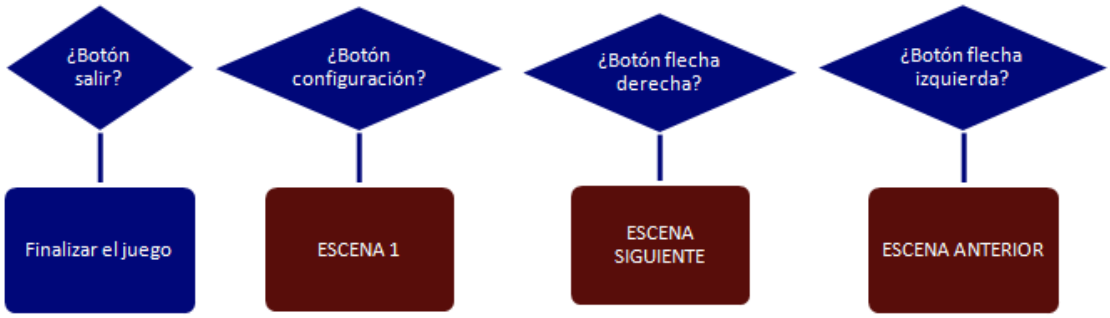


ESCENAS 2, 3 Y 4: Juego



ESCENA 5: Pantalla de resultados

Como se explicó en el apartado anterior, los botones de "salir", "configuración", "escena siguiente" y "escena anterior" están disponibles durante el juego y pueden ser pulsados en cualquier momento, si esto ocurre se interrumpirá el diagrama que esté actuando y ocurrirá lo siguiente:



Los scripts utilizados para realizar las acciones de los diagramas son los que se exponen a continuación. Se explicarán brevemente en el siguiente apartado (4.4.4.) y se puede encontrar el código completo en los anexos.

(1) GetInput

(2) Sliders

(3) ConfTeclas

(4) Colores

(5) Tecla_(cada tecla tendrá su propio nombre y código)

(6) Botones

(7) BarChart

4.4.4. DESCRIPCIÓN DE LOS SCRIPTS

A continuación se incluye una breve descripción de los scripts que se han utilizado para la elaboración del *serious game*. La lista se encuentra ordenada según su aparición en el juego. El código completo de cada uno de ellos se incluye en los anexos.

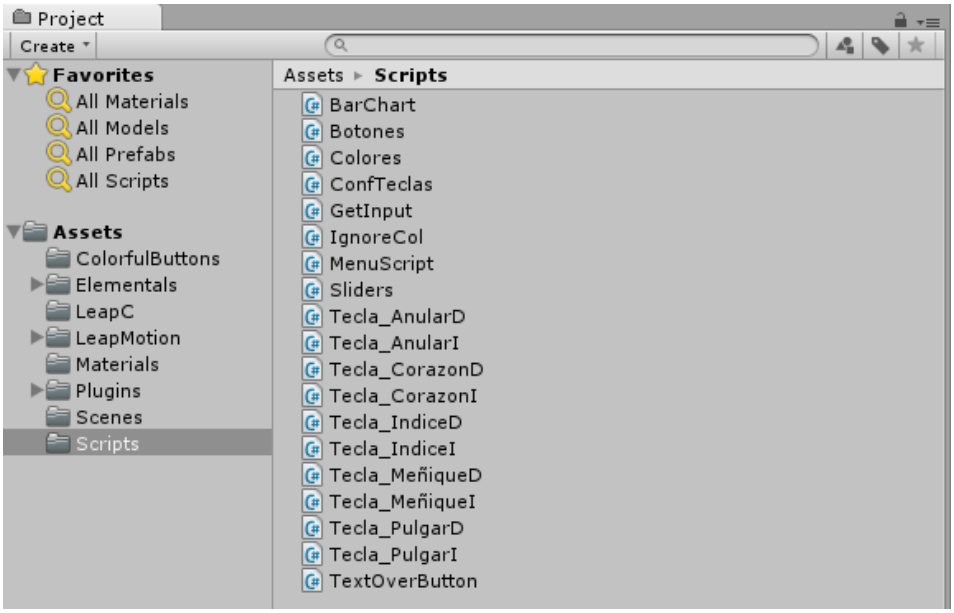


Figura 41: Listado de scripts en Unity.

GetInput

Se utiliza en la pantalla inicial para:

- Recoger la ruta donde se desea guardar el documento con los resultados.
- Recopilar e imprimir en dicho documento los datos del usuario que se introducen por teclado (nombre y apellido, mano afectada y motivo de la rehabilitación).
- Toma del ordenador la fecha y la hora en la que se está realizando la sesión y la anota en el documento.
- En el caso de que sea la primera vez que el paciente juega, se creará un archivo CSV que tendrá por nombre el nombre y apellidos introducidos. Este archivo CSV puede ser fácilmente importado a Excel.
- En cambio, si el paciente ya ha jugado alguna vez, el programa reconocerá el documento con su nombre y añadirá los resultados de la sesión junto con los anteriores.

MenuScript

Se usa en la primera escena para comenzar el juego, es decir, pasar a la siguiente pantalla, en el caso de que se presione el botón de play o finalizar el juego si se pulsa el botón de salir.

Sliders

Este script se encarga de regular el tamaño y posición de las teclas y de las manos virtuales en la escena 1. Puesto que se recurrirá a esta pantalla tanto antes de comenzar el juego como en cualquier momento en que el usuario desee variar algo del diseño, lo primero que hace el código es situar los elementos tal y como se colocaron previamente o mantenerlos en la posición inicial que viene por defecto.

Cuando el usuario haga uso de los deslizadores para adaptar las teclas y las manos a su gusto o a la necesidad de la terapia, el script colocará las teclas según la distancia entre ellas, su grosor y la altura de las manos requeridas. Esta información se guardará en

variables públicas para ser leídas por los scripts que manejan a las otras escenas y mantener el mismo diseño en ellas.

Este mismo script será el encargado de salir del juego, en el caso de que el usuario así lo requiera pulsando el botón de salir, y de cambiar de escena. En el caso de que se encuentre en la fase previa al juego, al pulsar la flecha hacia la derecha la siguiente pantalla será la escena 2 en que se realiza el juego con tan solo la mano derecha. Pero si nos encontramos en esta pantalla porque se ha pulsado el botón de configuración durante el juego, al pulsar la flecha hacia la derecha el programa recordará a través de la variable pública *level* (figura 42) desde qué escena (o nivel, como entiende Unity) fue llamado y volverá a él.

```
public void NextLevel()
{
    if (level == 0)
    {
        Application.LoadLevel(2);
    }
    else
    {
        Application.LoadLevel(level);
    }
}
```

Figura 42: Código para el cambio de escena.

ConfTeclas

El nombre de este script viene de "configurar las teclas". Será lo primero que se lea al iniciar cualquiera de las escenas de juego (2, 3 ó 4) para utilizar la distancia, grosor y altura de las teclas según se adaptaron en la pantalla de configuración. El programa reconocerá en qué nivel se encuentra y dependiendo de ello colocará centradas tan solo las cinco teclas de la derecha para jugar con la mano derecha, las cinco de la izquierda para jugar con la mano izquierda o las diez teclas para jugar con ambas manos.

Colores

Esta secuencia de comandos tiene la tarea de llevar a cabo el juego y recopilar los tiempos. Tras reconocer en que escena se encuentra, se ejecutarán cuatro series de

pulsaciones de la/s mano/s que correspondan, las dos primeras serán ordenadas como ya se explicó en el apartado 4.2. y las dos siguientes serán aleatorias.

Como se indicaba en el diagrama de flujo se realizarán las siguientes tareas:

- Se ilumina la falange distal del dedo para señalizar al jugador con cuál debe pulsar la tecla.
- Se habilita el sonido de la tecla de la que es el turno para que se pueda reproducir el sonido cuando sea pulsada.
- La tecla que corresponda se coloreará de negro para señalizar que es la que hay que presionar.

En este momento se iniciará un cronómetro y el código quedará a la espera de que la tecla sea pulsada. Cuando esto ocurra, se guardará el tiempo en la variable que le pertenezca y se reiniciará el cronómetro. Por último, se deshabilitará el sonido de esa tecla.

IgnoreCol

Su nombre hace referencia a "ignorar colisiones". Este script es el responsable de indicar entre qué componentes deben ignorarse las colisiones, es decir, entre todos los huesos de cada dedo y las teclas que no se correspondan.

En el ejemplo de la figura 43 se muestra el código que ordena que se ignore la colisión entre la tecla correspondiente al pulgar derecho (TPD) con los tres huesos de los dedos índice, corazón, anular y meñique, pero no del pulgar ya que esa colisión es la permitida.

```
//Tecla pulgar derecha con los dedos de la mano derecha

Physics.IgnoreCollision(TPD.GetComponent<Collider>(), Indice_bone1_D.GetComponent<Collider>(), true);
Physics.IgnoreCollision(TPD.GetComponent<Collider>(), Indice_bone2_D.GetComponent<Collider>(), true);
Physics.IgnoreCollision(TPD.GetComponent<Collider>(), Indice_bone3_D.GetComponent<Collider>(), true);
Physics.IgnoreCollision(TPD.GetComponent<Collider>(), Corazon_bone1_D.GetComponent<Collider>(), true);
Physics.IgnoreCollision(TPD.GetComponent<Collider>(), Corazon_bone2_D.GetComponent<Collider>(), true);
Physics.IgnoreCollision(TPD.GetComponent<Collider>(), Corazon_bone3_D.GetComponent<Collider>(), true);
Physics.IgnoreCollision(TPD.GetComponent<Collider>(), Anular_bone1_D.GetComponent<Collider>(), true);
Physics.IgnoreCollision(TPD.GetComponent<Collider>(), Anular_bone2_D.GetComponent<Collider>(), true);
Physics.IgnoreCollision(TPD.GetComponent<Collider>(), Anular_bone3_D.GetComponent<Collider>(), true);
Physics.IgnoreCollision(TPD.GetComponent<Collider>(), Meñique_bone1_D.GetComponent<Collider>(), true);
Physics.IgnoreCollision(TPD.GetComponent<Collider>(), Meñique_bone2_D.GetComponent<Collider>(), true);
Physics.IgnoreCollision(TPD.GetComponent<Collider>(), Meñique_bone3_D.GetComponent<Collider>(), true);
```

Figura 43: Código que permite que se ignoren las teclas con los dedos que no le corresponden.

Tecla_X

Existe un código con este nombre para cada tecla, siendo X el nombre del dedo de cada mano (por ejemplo, Tecla_PulgarD para el dedo pulgar derecho y Tecla_PulgarI para el dedo pulgar izquierdo). Al colisionar la tecla con su respectivo dedo se comprobará si el audio está habilitado y en caso afirmativo se reproducirá. Además, si la tecla es negra por ser su turno de ser pulsada, al presionarla volverá a su color blanco inicial para indicar que el ejercicio ya se ha realizado correctamente.

Botones

Es el encargado de detectar si alguno de los botones de la pantalla ("salir", "configuración", "siguiente escena" o "escena anterior") ha sido seleccionado por el usuario. En tal caso realizará la acción que le corresponda:

- Salir: de haberlos, guardará los resultados que se hayan realizado hasta el momento en el archivo y saldrá de la aplicación.
- Configuración: detectará en qué nivel se encuentra y lo guardará en la variable pública *level* que se mencionaba en la descripción de Sliders para saber a qué escena debe regresar.
- Escena siguiente o anterior: de haberlos, guardará los resultados que se hayan realizado hasta el momento en el archivo y pasará a la escena solicitada.

BarChart

Se calculará la media aritmética de los ocho tiempos empleados con cada dedo: los cuatro pertenecientes al ejercicio que se realiza con una mano más los cuatro empleados en el ejercicio con ambas manos. Si algún ejercicio no se realiza o no se termina, esos tiempos nulos serán descartados del cálculo de la media.

En la pantalla aparecerá un diagrama de barras donde cada barra de color pertenece a uno de los dedos de la mano izquierda o derecha y su altura corresponde al tiempo medio que se explicaba en el párrafo anterior.

Con este diagrama es fácil realizar una comparativa visual de con qué mano y con qué dedos le está resultando más difícil al paciente realizar los ejercicios.

TextOverButton

Se encarga de hacer aparecer bajo los botones, al pasar el ratón por encima, un texto explicativo de su función.

4.4. TRABAJOS PREVIOS

Con el mismo software y hardware de este proyecto se realizaron otros dos juegos que se quedaron en versiones iniciales pero sirvieron de ejemplo para que los terapeutas pudiesen ver el funcionamiento del Leap Motion y cómo se puede trabajar con él.

El primero de ellos trabaja los movimientos laterales y el alcance. En este juego aparecen unos cubos a distintas alturas y distancias y el paciente deberá ser capaz de tocarlos. Cada vez que se consiga tocar un cubo, este cambiará de color y caerá al suelo para indicar que se ha logrado. El paciente tendrá el reto de tirar todos los cubos poniendo a prueba el alcance y movimientos de su hombro.

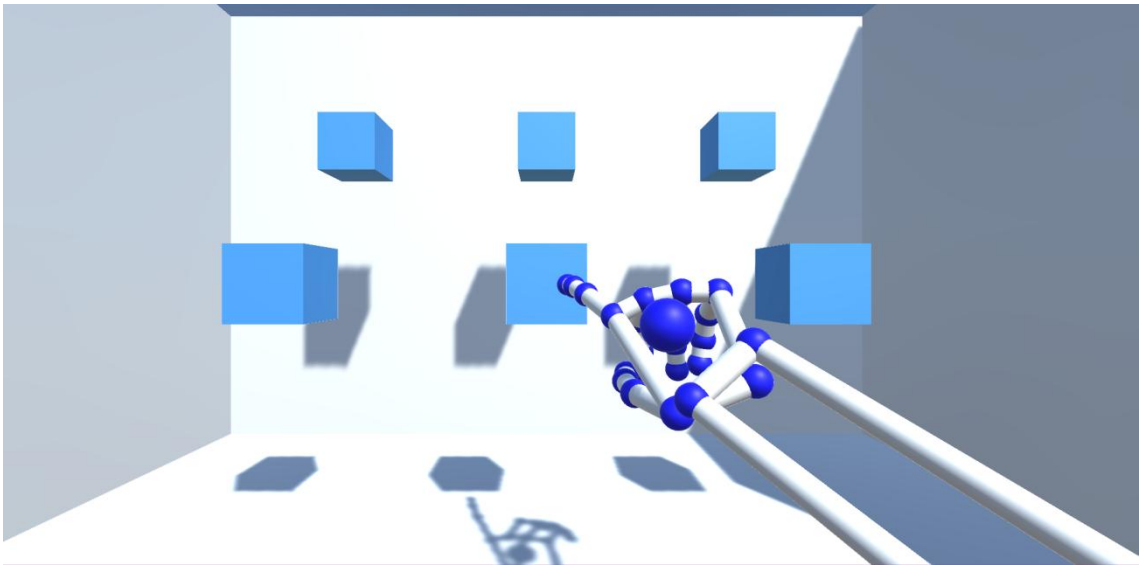


Figura 44: Juego de los cubos antes de tocar ninguno.

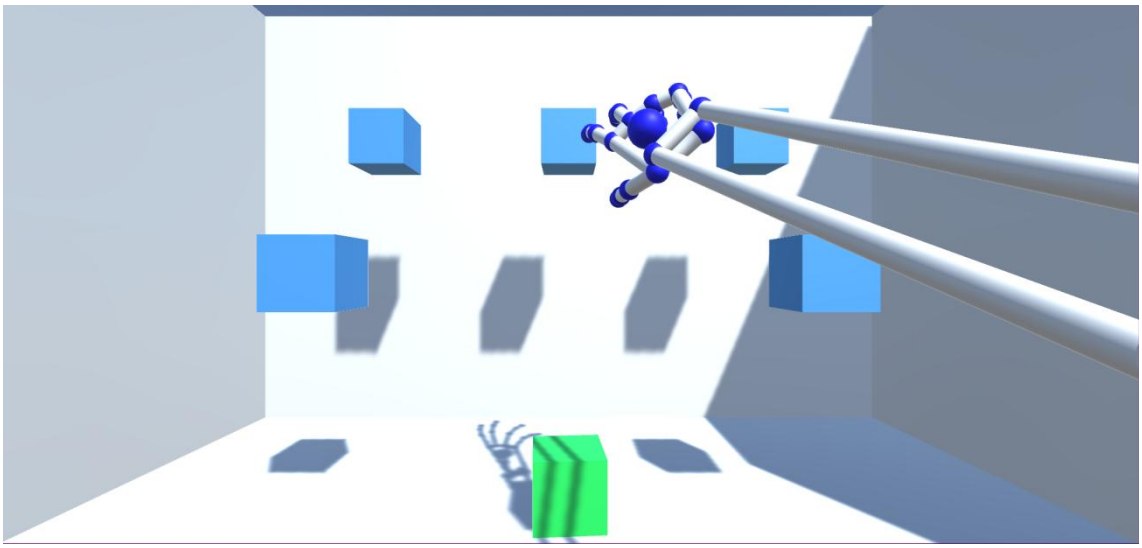


Figura 45: Juego de los cubos tras alcanzar uno de ellos.

El segundo juego lleva a la realidad virtual un ejercicio que se realiza en pacientes que padecen de negligencia visuoespacial. Estas personas no ven lo que se encuentra en el lado opuesto a la lesión cerebral, es por ello que en el ejercicio que aparece en la figura 46, donde se les pide cancelar todas las líneas que aparecen en la hoja, tan solo en uno de los lados se encuentran líneas tachadas ya que ignoran las que aparecen en el otro lado.

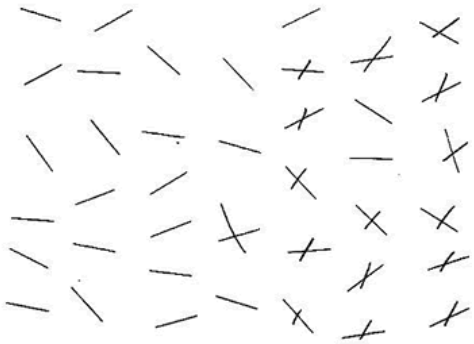


Figura 46: Test de cancelación de líneas.

Se ha creado un videojuego con este mismo ejercicio con la variante de que se pueden colocar las líneas a distintas profundidades. El paciente deberá tocar todas aquellas que estén en su campo de visión y se irá llevando una cuenta (véanse los marcadores en la parte superior de la pantalla de las figura 47 y 48) de cuántas líneas se han tirado de cada lado.

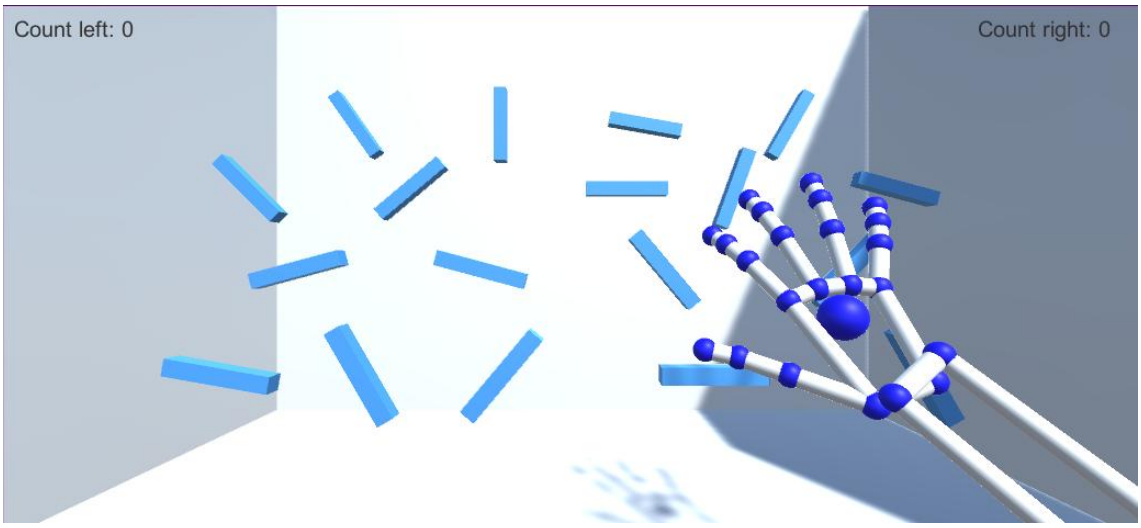


Figura 47: Test de cancelación antes de comenzar a jugar.

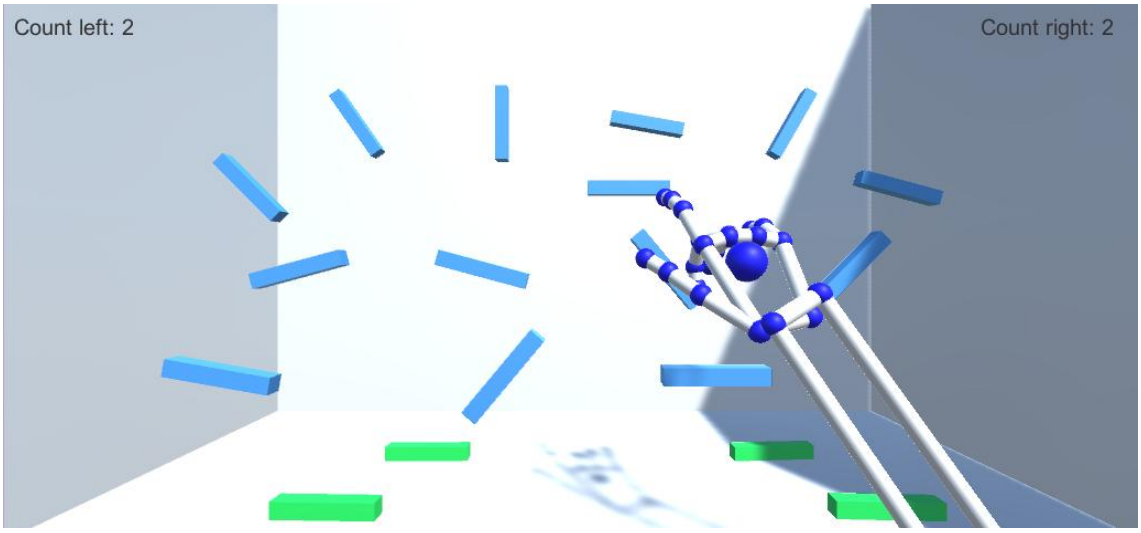


Figura 48: Test de cancelación tras tirar dos líneas de cada lado.

Las líneas solo reaccionarán al colisionar con la punta del dedo índice para evitar que puedan ser derrumbadas sin querer con cualquier otra parte de la mano o brazo al intentar alcanzar otra más lejana.

CAPÍTULO 5.

CONCLUSIONES Y TRABAJOS FUTUROS

En este capítulo se exponen los objetivos que se han alcanzado tras la realización del proyecto, las conclusiones obtenidas, posibles mejoras que se le podrían realizar y cómo se podría seguir desarrollando en el futuro.

5.1. CONCLUSIONES

El objetivo principal de este proyecto era la creación de un *serious game* enfocado a la rehabilitación de las manos con la finalidad de mejorar la disociación de los dedos y la coordinación bimanual. Este objetivo se ha alcanzado desarrollando un juego que requiere que el sujeto mueva los dedos de forma independiente y que, en cierto punto, incrementando la dificultad del ejercicio, sea necesario involucrar las dos manos para lograr la meta.

El diseño del videojuego está basado en el modelo de un piano de diez teclas, correspondiéndose cada tecla a un dedo de la mano, y que deberán ser pulsadas según se vaya indicando en el juego. Además, es personalizable al incluir opciones de ajuste de tamaño y posición de teclas y altura de manos, de acuerdo a la zona más apropiada de detección, y donde el paciente se sienta cómodo o donde el terapeuta considere beneficioso para la rehabilitación. Esta flexibilidad hace que el videojuego presentado se pueda utilizar para tratar distintas patologías y en diversos tratamientos de rehabilitación y, por tanto, llegar a un elevado número de usuarios.

Otro de los objetivos fundamentales era poder hacer un seguimiento de los movimientos de las manos del paciente y obtener información relevante a partir de ellos. Gracias al Leap Motion esto ha sido posible. Puesto que el software de dicho dispositivo reconoce manos, dedos y huesos, se ha podido programar el juego de forma que se fuerce al paciente a pulsar cada tecla del piano con el dedo que le

corresponde ignorando cualquier colisión con el resto de dedos, así deberán moverse los dedos de forma independiente.

Todos los resultados de los ejercicios serán guardados en un archivo CSV correspondiente a cada paciente. Este archivo puede ser abierto fácilmente con Excel, lo que facilita la evaluación de los resultados y progresos del paciente por parte del terapeuta, encontrando toda su documentación e información de las sesiones en dicho archivo.

Por otro lado, la introducción de nuevas tecnologías y dispositivos como los utilizados en este trabajo, han permitido generar escenarios virtuales con los que se introducen factores de motivación, entretenimiento e incluso competitividad, dentro de un proceso de rehabilitación física en este caso. Esta nueva metodología permite mejorar los procesos de terapia tradicional. Para evaluar en qué medida mejora la efectividad del tratamiento y aceptación por parte de los usuarios, el videojuego debe ser evaluado con usuarios reales.

Por último, gracias al código abierto de Unity y al precio económico y pequeñas dimensiones del Leap Motion, se ha podido crear un *serious game* asequible y que puede ser ejecutado desde cualquier lugar, tanto en centros de rehabilitación como desde casa.

5.2. TRABAJOS FUTUROS

Es necesario que un equipo de profesionales evalúen nuevamente el *serious game* tras su finalización para poder sacar el máximo partido de él. Es posible extraer más información de las manos y dedos, y bajo la petición de un terapeuta sería factible incluir la más relevante en el videojuego. Asimismo, se podría alterar el programa de forma que tanto el terapeuta como el propio paciente puedan personalizar más variables del juego además del tamaño y posición de las teclas y las manos, como por ejemplo, seleccionar el nivel de dificultad.

Al tratarse de un software de código abierto, estas modificaciones pueden ser llevadas a cabo por cualquier persona con conocimientos de programación y con acceso a la aplicación.

Finalmente, se deberían realizar ensayos con pacientes reales probando este videojuego y así obtener un feedback desde el punto de vista de los usuarios y a partir de ello realizar modificaciones que los involucren más.

PRESUPUESTO

El presupuesto global asociado al proyecto realizado se desglosa a continuación:

UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

PRESUPUESTO DE PROYECTO

1.- Autor: Cynthia Mª Cubeiro Tuimil

2.- Departamento: Ingeniería de Sistemas y Automática

3.- Descripción del Proyecto: Implementación de aplicaciones *serious games* para la evaluación de destreza manual

- Titulo **Ingeniero Industrial**
- Duración (meses) **8**

Tasa de costes Indirectos: **15%**

4.- Presupuesto total del Proyecto:

Presupuesto (sin IVA) 25.211,91 €
Presupuesto (con IVA) 30.506,42 €

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F. (no rellenar - solo a titulo informativo)	Categoría	Dedicación [a]	Coste mes	Coste	Firma de conformidad
Cubeiro Tuimil, Cynthia Mª		Ingeniero Senior Ingeniero	8	4.289,54 € 2.694,39 €	21.555,12 €	
				Total	21.555,12 €	

[a] 1 persona = 131,25 h/mes

EQUIPOS

Descripción	Coste	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d]
Leap Motion	57,84 €	100	8	60	7,71 €
Adaptador Leap Motion para VR	16,52 €	50	8	60	1,10 €
Oculus Rift	612,40 €	50	8	60	40,83 €
MSI Dominator Pro	2.389,83 €	100	8	60	318,64 €
				Total	368,28

[d] Fórmula de cálculo de la Amortización:

$$\frac{A}{B} \text{xCxD}$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

6.- Resumen de costes

Descripción	Presupuesto Costes Totales
Personal	21.555,12 €
Amortización	368,28 €
Total costes directos	21.923,40 €
Costes indirectos	3.288,51 €
Total	**25.211,91 €**

REFERENCIAS

[1] Laboratorio de Análisis del Movimiento, Biomecánica, Ergonomía y Control Motor (LAMBECOM), [Último acceso el 15-09-2016]. URL: https://www.urjc.es/actualidad/noticias/432-laboratorio-de-analisis-del-movimiento-biomecanica-ergonomia-y-control-motor

[2] Andrés Navarro, Juan Vicente Pradilla y Octavio Ríos: "Open Source 3D Games Engines for Serious Games Modeling", Prof. Catalin Alexandru (Ed.), 2012, ISBN: 978-953-51- 0012-6, InTech

[3] RoboHealth-A, 2014, [Último acceso el 15-09-2016]. URL: http://roboticslab.uc3m.es/roboticslab/project/robohealth

[4] Centro de Referencia Estatal de Autonomía Personal y Ayudas Técnicas (Ceapat), [Último acceso el 04-10-2016]. URL: http://www.ceapat.es/InterPresent1/groups/imserso/documents/binario/im_032579.pdf

[5] "Hemiparesis Assessment tools", en Wikipedia, Wikimedia Foundation, 2016, [Último acceso el 14-09-2016]. URL: https://en.wikipedia.org/wiki/Hemiparesis#Assessment_tools

[6] Practicasfisio, WordPress, 2015, [Último acceso el 14-09-2016]. URL: https://practicasfisio.wordpress.com/2015/04/15/ese-mundo-maravilloso-llamado-escalas-de-fisioterapia/

[7] "Assessments", en Stroke Engine, 2016, [Último acceso el 14-09-2016]. URL: http://www.strokengine.ca/assess/

[8] "Fugl-Meyer Assessment", en Stroke Engine, 2016, [Último acceso el 14-09-2016]. URL: http://www.strokengine.ca/indepth/fma_indepth/

[9] Virgil Mathiowetz, Gloria Volland, Nancy Kashman y Karen Weber: "Adult Norms for the Box and Block Test of Manual Dexterity", American Journal of Occupational Therapy, Vol. 39, junio 1985.

[10] "Purdue Pegboard Test", en Wikipedia, Wikimedia Foundation, 2016, [Último acceso el 14-09-2016]. URL: https://en.wikipedia.org/wiki/Purdue_Pegboard_Test

[11] Huiyu Zhou, Huosheng Hu: "Human motion tracking for rehabilitation - A survey", Elsevier, Biomedical Signal Processing and Control, Vol. 3, enero 2008.

[12] Hossein Mousavi Hondori y Maryam Khademi: "A Review on Technical and Clinical Impact of Microsoft Kinect on Physical Therapy and Rehabilitation", Hindawi Publishing Corporation Journal of Medical Engineering, Vol. 2014, diciembre 2014.

[13] Alessandro de Mauro: "Virtual Reality Based Rehabilitation and Game Technology", eHealth and Biomedical Applications, Vicomtech, 2011.

[14] Patrice L Weiss, Debbie Rand, Noomi Katz y Rachel Kizony: "Video capture virtual reality as a flexible and effective rehabilitation tool", Journal of NeuroEngineering and Rehabilitation, diciembre 2004.

[15] Marco Iosa, Giovanni Morone, Augusto Fusco, Marcello Castagnoli, Francesca Romana Fusco, Luca Pratesi, Stefano Paolucci: "Leap motion controlled videogame-based therapy for rehabilitation of elderly patients with subacute stroke: a feasibility pilot study", Topics in Stroke Rehabilitation, Vol. 22, 2015.

[16] J H Crosbie, S Lennon, M C McGoldrick, M D J McNeill, J W Burke y S M McDonough: "Virtual reality in the rehabilitation of the upper limb alter hemiplegic stroke: a randomised pilot study", ICDVRAT, 2008.

[17] Manjuladevi Kuttuva, Rares Boian, Alma Merians, Griogore Burdea, Mourad Bouzit, Jeffrey Lewis y Devin Fensterheim: "The Rutgers Arm: An Upper-Extremity Rehabilitation System in Virtual Reality", IWVR, 2005.

[18] VirtualRehab, 2014, [Último acceso el 08-09-2016]. URL: http://www.virtualrehab.info/

[19] Visual Touch Therapy, 2016, [Último acceso el 08-09-2016]. URL: http://www.visualtouchtherapy.com/

[20] "Healthcare", en Leap Motion, [Último acceso el 21-09-2016]. URL: https://www.leapmotion.com/solutions/healthcare

[21] "5 medical and assistive technologies being transformed with Leap Motion", en Leap Motion Blog, 2015, [Último acceso el 21-09-2016]. URL: http://blog.leapmotion.com/5-medical-and-assistive-technologies-being-transformed-with-leap-motion-tech/

[22] TedCas, [Último acceso el 21-09-2016]. URL: http://www.tedcas.com/

[23] MotionSavvy, [Último acceso el 21-09-2016]. URL: http://www.motionsavvy.com/

[24] Vivid Vision, 2015, [Último acceso el 21-09-2016]. URL: https://www.seevividly.com/

[25] Burke Rehabilitation Hospital, [Último acceso el 21-09-2016]. URL: http://www.burke.org/research

[26] TenTon Raygun, [Último acceso el 21-09-2016]. URL: http://tentonraygun.com/

[27] "Company facts", en Unity, [Último acceso el 23-09-2016]. URL: https://unity3d.com/public-relations

[28] "Multiplataforma", en Unity, [Último acceso el 23-09-2016]. URL: https://unity3d.com/es/unity/multiplatform

[29] "Unity", en Wikipedia, Wikimedia Foundation, 2016, [Último acceso el 23-09-2016]. URL: https://es.wikipedia.org/wiki/Unity_(software)

[30] "Documentation, Unity scripting languages and you", en el blog de Unity, Unity, 2014, [Último acceso el 23-09-2016]. URL:

https://blogs.unity3d.com/2014/09/03/documentation-unity-scripting-languages-and-you/

[31] Unity Asset Store, Unity, [Último acceso el 23-09-2016]. URL: https://www.assetstore.unity3d.com/en/#!/home

[32] J. Godlove, A. Tsu, K. Ganguly: "Comparison of low-cost hand-monitoring devices for motor rehabilitation", UCSF

[33] "How does the Leap Motion Controller work?", en el blog de Leap Motion, 2014, [Último acceso el 26-09-2016]. URL: http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/

[34] Maryam Khademi, Hossein Mousavi Hondori, Alison McKenzie, Lucy Dodakian, Cristina V. Lopes y Steven C. Cramer: "Free-Hand Interaction with Leap Motion Controller for Stroke Rehabilitation", CHI 2014.

[35] "Developer documentation", en Leap Motion, [Último acceso el 26-09-2016]. URL: https://developer.leapmotion.com/documentation/unity/index.html

ANEXOS

ANEXO 1. CÓDIGO COMPLETO DE LOS SCRIPTS

GetInput

```
using UnityEngine;
using System.Collections;
using System.IO;
using System.Globalization;

public class GetInput : MonoBehaviour {

    public static string saveFilePath;
    public static string ruta;
    string nombre;
    string mano;
    string lesion;

    public void GetFilePath (string path)    {
        saveFilePath = path;    }

    public void GetName(string name)    {
        ruta = saveFilePath + "/" + name + ".csv";
        nombre = name;    }

    public void GetHand(string hand)    {
        mano = hand;    }

    public void GetInjury(string injury)    {
        lesion = injury;      }

    public void CreateFile()    {
        if (!File.Exists(ruta)) {
            using (StreamWriter writer = File.CreateText(ruta))          {
                string sLine = "NOMBRE: " + nombre;
                writer.WriteLine(sLine);
                System.DateTime localDate = System.DateTime.Now;
                var culture = new CultureInfo("es-ES");
                writer.WriteLine("FECHA: " + localDate.ToString(culture));
                string sLine2 = "MANO LESIONADA: " + mano;
                writer.WriteLine(sLine2);
                string sLine3 = "MOTIVO DE LA REHABILITACION: " + lesion;
                writer.WriteLine(sLine3);
                writer.WriteLine("\nT1 es el tiempo empleado en pulsar durante la primera serie en orden \nT2 es el tiempo empleado
                    en pulsar durante la segunda serie en orden \nT3 es el tiempo empleado en pulsar durante la primera serie aleatoria
                    \nT4 es el tiempo empleado en pulsar durante la segunda serie aleatoria" + ", T1 (s), T2 (s), T3 (s), T4 (s)");          }
        }  else      {
            using (StreamWriter writer = File.AppendText(ruta))       {
                string sLine = "\n\nNOMBRE: " + nombre;
                writer.WriteLine(sLine);
                System.DateTime localDate = System.DateTime.Now;
                var culture = new CultureInfo("es-ES");
                writer.WriteLine("FECHA: " + localDate.ToString(culture));
                string sLine2 = "MANO LESIONADA: " + mano;
                writer.WriteLine(sLine2);
                string sLine3 = "MOTIVO DE LA REHABILITACION: " + lesion;
                writer.WriteLine(sLine3);          }
        }
    }
}
```

MenuScript

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class MenuScript : MonoBehaviour {
    public Canvas quitMenu;
    public Button startText;
    public Button exitText;

    void Start () {
        quitMenu = quitMenu.GetComponent<Canvas>();
        startText = startText.GetComponent<Button>();
        exitText = exitText.GetComponent<Button>();
        quitMenu.enabled = false;    }

   public void ExitPress() {
        quitMenu.enabled = true;
        startText.enabled = false;
        exitText.enabled = false;    }

    public void ExitGame()   {
        Application.Quit();    }

    public void NoPress()    {
        quitMenu.enabled = false;
        startText.enabled = true;
        exitText.enabled = true;    }

    public void StartLevel()    {
        Application.LoadLevel(1);    }
}
```

Sliders

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class Sliders : MonoBehaviour {
    public GameObject HandCont;
    public GameObject TPD;
    public GameObject TID;
    public GameObject TCD;
    public GameObject TAD;
    public GameObject TMD;
    public GameObject TPI;
    public GameObject TII;
    public GameObject TCI;
    public GameObject TAI;
    public GameObject TMI;
    public Canvas quitMenu;
    public Button next;
    public Button exit;
    public static float altura;
    public static float distancia;
    public static float grosor;
    public static int level;
    public Slider SliderAlt;
    public Slider SliderDist;
    public Slider SliderGros;

    void Start()    {
        quitMenu = quitMenu.GetComponent<Canvas>();
        next = next.GetComponent<Button>();
        exit = exit.GetComponent<Button>();
```

```
quitMenu.enabled = false;
if (altura != 0)      {
   Vector3 altInicial = HandCont.transform.position;
   altInicial.y = altura;
   HandCont.transform.position = altInicial;
   SliderAlt.value = altura;
}  else      {
   altura = SliderAlt.value;      }
if (distancia != 0)      {
   Vector3 posTPD = TPD.transform.position;
   Vector3 posTID = TID.transform.position;
   Vector3 posTCD = TCD.transform.position;
   Vector3 posTAD = TAD.transform.position;
   Vector3 posTMD = TMD.transform.position;
   posTID.x = posTPD.x + grosor + distancia;
   TID.transform.position = posTID;
   posTCD.x = posTPD.x + 2 * grosor + 2 * distancia;
   TCD.transform.position = posTCD;
   posTAD.x = posTPD.x + 3 * grosor + 3 * distancia;
   TAD.transform.position = posTAD;
   posTMD.x = posTPD.x + 4 * grosor + 4 * distancia;
   TMD.transform.position = posTMD;
   SliderDist.value = distancia;
   Vector3 posTPI = TPI.transform.position;
   Vector3 posTII = TII.transform.position;
   Vector3 posTCI = TCI.transform.position;
   Vector3 posTAI = TAI.transform.position;
   Vector3 posTMI = TMI.transform.position;
   posTII.x = posTPI.x - grosor - distancia;
   TII.transform.position = posTII;
   posTCI.x = posTPI.x - 2 * grosor - 2 * distancia;
   TCI.transform.position = posTCI;
   posTAI.x = posTPI.x - 3 * grosor - 3 * distancia;
   TAI.transform.position = posTAI;
   posTMI.x = posTPI.x - 4 * grosor - 4 * distancia;
   TMI.transform.position = posTMI;
}  else      {
   distancia = SliderDist.value;      }
if (grosor != 0) {
   Vector3 tamTPD = TPD.transform.localScale;
   Vector3 tamTID = TID.transform.localScale;
   Vector3 tamTCD = TCD.transform.localScale;
   Vector3 tamTAD = TAD.transform.localScale;
   Vector3 tamTMD = TMD.transform.localScale;
   Vector3 tamTPI = TPI.transform.localScale;
   Vector3 tamTII = TII.transform.localScale;
   Vector3 tamTCI = TCI.transform.localScale;
   Vector3 tamTAI = TAI.transform.localScale;
   Vector3 tamTMI = TMI.transform.localScale;
   SliderGros.value = grosor;
   tamTPD.x = grosor;
   TPD.transform.localScale = tamTPD;
   tamTID.x = grosor;
   TID.transform.localScale = tamTID;
   tamTCD.x = grosor;
   TCD.transform.localScale = tamTCD;
   tamTAD.x = grosor;
   TAD.transform.localScale = tamTAD;
   tamTMD.x = grosor;
   TMD.transform.localScale = tamTMD;
   tamTPI.x = grosor;
   TPI.transform.localScale = tamTPI;
   tamTII.x = grosor;
   TII.transform.localScale = tamTII;
   tamTCI.x = grosor;
   TCI.transform.localScale = tamTCI;
   tamTAI.x = grosor;
   TAI.transform.localScale = tamTAI;
   tamTMI.x = grosor;
   TMI.transform.localScale = tamTMI;
}  else      {
```

```
        grosor= SliderGros.value;       }
}

public void ExitPress()    {
    quitMenu.enabled = true;
    next.enabled = false;
    exit.enabled = false;    }

public void NoPress()    {
    quitMenu.enabled = false;
    next.enabled = true;
    exit.enabled = true;    }

public void NextLevel()    {
    if (level == 0)       {
        Application.LoadLevel(2);
    }  else       {
        Application.LoadLevel(level);       }
}

public void ExitGame()    {
    Application.Quit();    }

public void Slider_Altura(float newValue)    {
    Vector3 pos = HandCont.transform.position;
    altura = newValue;
    pos.y = newValue;
    HandCont.transform.position = pos;    }

public void Slider_Distancia(float newValue)    {
    distancia = newValue;
    Vector3 posTPD = TPD.transform.position;
    Vector3 posTID = TID.transform.position;
    Vector3 posTCD = TCD.transform.position;
    Vector3 posTAD = TAD.transform.position;
    Vector3 posTMD = TMD.transform.position;
    Vector3 tamTPD = TPD.transform.localScale;
    posTID.x = posTPD.x + tamTPD.x + newValue;
    TID.transform.position = posTID;
    posTCD.x = posTPD.x + 2 * tamTPD.x + 2 * newValue;
    TCD.transform.position = posTCD;
    posTAD.x = posTPD.x + 3 * tamTPD.x + 3 * newValue;
    TAD.transform.position = posTAD;
    posTMD.x = posTPD.x + 4 * tamTPD.x + 4 * newValue;
    TMD.transform.position = posTMD;
    Vector3 posTPI = TPI.transform.position;
    Vector3 posTII = TII.transform.position;
    Vector3 posTCI = TCI.transform.position;
    Vector3 posTAI = TAI.transform.position;
    Vector3 posTMI = TMI.transform.position;
    Vector3 tamTPI = TPI.transform.localScale;
    posTII.x = posTPI.x - tamTPI.x - newValue;
    TII.transform.position = posTII;
    posTCI.x = posTPI.x - 2 * tamTPI.x - 2 * newValue;
    TCI.transform.position = posTCI;
    posTAI.x = posTPI.x - 3 * tamTPI.x - 3 * newValue;
    TAI.transform.position = posTAI;
    posTMI.x = posTPI.x - 4 * tamTPI.x - 4 * newValue;
    TMI.transform.position = posTMI;    }

public void Slider_Grosor(float newValue)    {
    Vector3 tamTPD = TPD.transform.localScale;
    Vector3 tamTID = TID.transform.localScale;
    Vector3 tamTCD = TCD.transform.localScale;
    Vector3 tamTAD = TAD.transform.localScale;
    Vector3 tamTMD = TMD.transform.localScale;
    Vector3 tamTPI = TPI.transform.localScale;
    Vector3 tamTII = TII.transform.localScale;
    Vector3 tamTCI = TCI.transform.localScale;
    Vector3 tamTAI = TAI.transform.localScale;
    Vector3 tamTMI = TMI.transform.localScale;
```

```
    Vector3 posTPD = TPD.transform.position;
    Vector3 posTID = TID.transform.position;
    Vector3 posTCD = TCD.transform.position;
    Vector3 posTAD = TAD.transform.position;
    Vector3 posTMD = TMD.transform.position;
    Vector3 posTPI = TPI.transform.position;
    Vector3 posTII = TII.transform.position;
    Vector3 posTCI = TCI.transform.position;
    Vector3 posTAI = TAI.transform.position;
    Vector3 posTMI = TMI.transform.position;
      grosor = newValue;
      tamTPD.x = newValue;
      TPD.transform.localScale = tamTPD;
      tamTID.x = newValue;
      TID.transform.localScale = tamTID;
      tamTCD.x = newValue;
      TCD.transform.localScale = tamTCD;
      tamTAD.x = newValue;
      TAD.transform.localScale = tamTAD;
      tamTMD.x = newValue;
      TMD.transform.localScale = tamTMD;
      tamTPI.x = newValue;
      TPI.transform.localScale = tamTPI;
      tamTII.x = newValue;
      TII.transform.localScale = tamTII;
      tamTCI.x = newValue;
      TCI.transform.localScale = tamTCI;
      tamTAI.x = newValue;
      TAI.transform.localScale = tamTAI;
      tamTMI.x = newValue;
      TMI.transform.localScale = tamTMI;
      //Para colocar las teclas manteniendo la distancia
      posTID.x = posTPD.x + grosor + distancia;
      TID.transform.position = posTID;
      posTCD.x = posTPD.x + 2 * grosor + 2 * distancia;
      TCD.transform.position = posTCD;
      posTAD.x = posTPD.x + 3 * grosor + 3 * distancia;
      TAD.transform.position = posTAD;
      posTMD.x = posTPD.x + 4 * grosor + 4 * distancia;
      TMD.transform.position = posTMD;
      posTII.x = posTPI.x - grosor - distancia;
      TII.transform.position = posTII;
      posTCI.x = posTPI.x - 2 * grosor - 2 * distancia;
      TCI.transform.position = posTCI;
      posTAI.x = posTPI.x - 3 * grosor - 3 * distancia;
      TAI.transform.position = posTAI;
      posTMI.x = posTPI.x - 4 * grosor - 4 * distancia;
      TMI.transform.position = posTMI;    }
}
```

ConfTeclas

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class ConfTeclas : MonoBehaviour {
   public GameObject TPD;
   public GameObject TID;
   public GameObject TCD;
   public GameObject TAD;
   public GameObject TMD;
   public GameObject TPI;
   public GameObject TII;
   public GameObject TCI;
   public GameObject TAI;
   public GameObject TMI;
   public GameObject HandCont;
```

```
  void Start() {
    //MANO DERECHA
    if (Application.loadedLevel == 2)      {
      if (Sliders.altura != 0) {
        //porque si es igual a 0 significa que el deslizador correspondiente a la altura no se ha movido y, por tanto, se
mantiene la altura que venía por defecto
        Vector3 altura = HandCont.transform.position;
        altura.y = Sliders.altura;
        HandCont.transform.position = altura;        }
      if (Sliders.distancia != 0) {
        Vector3 posTPD = TPD.transform.position;
        Vector3 posTID = TID.transform.position;
        Vector3 posTCD = TCD.transform.position;
        Vector3 posTAD = TAD.transform.position;
        Vector3 posTMD = TMD.transform.position;
        posTCD.x = -(Sliders.grosor / 2);
        TCD.transform.position = posTCD;
        posTID.x = posTCD.x - Sliders.distancia - Sliders.grosor;
        TID.transform.position = posTID;
        posTPD.x = posTCD.x - 2 * Sliders.distancia - 2 * Sliders.grosor;
        TPD.transform.position = posTPD;
        posTAD.x = posTCD.x + Sliders.distancia + Sliders.grosor;
        TAD.transform.position = posTAD;
        posTMD.x = posTCD.x + 2 * Sliders.distancia + 2 * Sliders.grosor;
        TMD.transform.position = posTMD;        }
      if (Sliders.grosor != 0) {
        Vector3 tamTPD = TPD.transform.localScale;
        Vector3 tamTID = TID.transform.localScale;
        Vector3 tamTCD = TCD.transform.localScale;
        Vector3 tamTAD = TAD.transform.localScale;
        Vector3 tamTMD = TMD.transform.localScale;
        tamTPD.x = Sliders.grosor;
        TPD.transform.localScale = tamTPD;
        tamTID.x = Sliders.grosor;
        TID.transform.localScale = tamTID;
        tamTCD.x = Sliders.grosor;
        TCD.transform.localScale = tamTCD;
        tamTAD.x = Sliders.grosor;
        TAD.transform.localScale = tamTAD;
        tamTMD.x = Sliders.grosor;
        TMD.transform.localScale = tamTMD;        }
    }
    //MANO IZQUIERDA
    if (Application.loadedLevel == 3)      {
      if (Sliders.altura != 0)        {
        Vector3 altura = HandCont.transform.position;
        altura.y = Sliders.altura;
        HandCont.transform.position = altura;        }
      if (Sliders.distancia != 0)        {
        Vector3 posTPI = TPI.transform.position;
        Vector3 posTII = TII.transform.position;
        Vector3 posTCI = TCI.transform.position;
        Vector3 posTAI = TAI.transform.position;
        Vector3 posTMI = TMI.transform.position;
        posTCI.x = -(Sliders.grosor / 2);
        TCI.transform.position = posTCI;
        posTAI.x = posTCI.x - Sliders.distancia - Sliders.grosor;
        TAI.transform.position = posTAI;
        posTMI.x = posTCI.x - 2 * Sliders.distancia - 2 * Sliders.grosor;
        TMI.transform.position = posTMI;
        posTII.x = posTCI.x + Sliders.distancia + Sliders.grosor;
        TII.transform.position = posTII;
        posTPI.x = posTCI.x + 2 * Sliders.distancia + 2 * Sliders.grosor;
        TPI.transform.position = posTPI;        }
      if (Sliders.grosor != 0)        {
        Vector3 tamTPI = TPI.transform.localScale;
        Vector3 tamTII = TII.transform.localScale;
        Vector3 tamTCI = TCI.transform.localScale;
        Vector3 tamTAI = TAI.transform.localScale;
        Vector3 tamTMI = TMI.transform.localScale;
        tamTPI.x = Sliders.grosor;
```

```
        TPI.transform.localScale = tamTPI;
        tamTII.x = Sliders.grosor;
        TII.transform.localScale = tamTII;
        tamTCI.x = Sliders.grosor;
        TCI.transform.localScale = tamTCI;
        tamTAI.x = Sliders.grosor;
        TAI.transform.localScale = tamTAI;
        tamTMI.x = Sliders.grosor;
        TMI.transform.localScale = tamTMI;          }
    }
    //AMBAS MANOS
    if (Application.loadedLevel == 4)       {
      if (Sliders.altura != 0)          {
        Vector3 altura = HandCont.transform.position;
        altura.y = Sliders.altura;
        HandCont.transform.position = altura;          }
      if (Sliders.distancia != 0)          {
        Vector3 posTPD = TPD.transform.position;
        Vector3 posTID = TID.transform.position;
        Vector3 posTCD = TCD.transform.position;
        Vector3 posTAD = TAD.transform.position;
        Vector3 posTMD = TMD.transform.position;
        posTID.x = posTPD.x + Sliders.grosor + Sliders.distancia;
        TID.transform.position = posTID;
        posTCD.x = posTPD.x + 2 * Sliders.grosor + 2 * Sliders.distancia;
        TCD.transform.position = posTCD;
        posTAD.x = posTPD.x + 3 * Sliders.grosor + 3 * Sliders.distancia;
        TAD.transform.position = posTAD;
        posTMD.x = posTPD.x + 4 * Sliders.grosor + 4 * Sliders.distancia;
        TMD.transform.position = posTMD;
        Vector3 posTPI = TPI.transform.position;
        Vector3 posTII = TII.transform.position;
        Vector3 posTCI = TCI.transform.position;
        Vector3 posTAI = TAI.transform.position;
        Vector3 posTMI = TMI.transform.position;
        posTII.x = posTPI.x - Sliders.grosor - Sliders.distancia;
        TII.transform.position = posTII;
        posTCI.x = posTPI.x - 2 * Sliders.grosor - 2 * Sliders.distancia;
        TCI.transform.position = posTCI;
        posTAI.x = posTPI.x - 3 * Sliders.grosor - 3 * Sliders.distancia;
        TAI.transform.position = posTAI;
        posTMI.x = posTPI.x - 4 * Sliders.grosor - 4 * Sliders.distancia;
        TMI.transform.position = posTMI;          }
      if (Sliders.grosor != 0)          {
        Vector3 tamTPD = TPD.transform.localScale;
        Vector3 tamTID = TID.transform.localScale;
        Vector3 tamTCD = TCD.transform.localScale;
        Vector3 tamTAD = TAD.transform.localScale;
        Vector3 tamTMD = TMD.transform.localScale;
        Vector3 tamTPI = TPI.transform.localScale;
        Vector3 tamTII = TII.transform.localScale;
        Vector3 tamTCI = TCI.transform.localScale;
        Vector3 tamTAI = TAI.transform.localScale;
        Vector3 tamTMI = TMI.transform.localScale;
        tamTPD.x = Sliders.grosor;
        TPD.transform.localScale = tamTPD;
        tamTID.x = Sliders.grosor;
        TID.transform.localScale = tamTID;
        tamTCD.x = Sliders.grosor;
        TCD.transform.localScale = tamTCD;
        tamTAD.x = Sliders.grosor;
        TAD.transform.localScale = tamTAD;
        tamTMD.x = Sliders.grosor;
        TMD.transform.localScale = tamTMD;
        tamTPI.x = Sliders.grosor;
        TPI.transform.localScale = tamTPI;
        tamTII.x = Sliders.grosor;
        TII.transform.localScale = tamTII;
        tamTCI.x = Sliders.grosor;
        TCI.transform.localScale = tamTCI;
        tamTAI.x = Sliders.grosor;
```

```
            TAI.transform.localScale = tamTAI;
            tamTMI.x = Sliders.grosor;
            TMI.transform.localScale = tamTMI;          }
        }
    }
}
```

Colores

Debido a que este script tiene muchas líneas de código, en este anexo se incluye tan solo la parte correspondiente a la escena donde se realiza el ejercicio con ambas manos, siendo el propio para cada mano individualmente similar pero incluyendo tan solo las series con una mano y utilizando la función de crear números aleatorios valores entre 1 y 5.

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
using System.Diagnostics;
using System.IO;

public class Colores : MonoBehaviour {
    public GameObject TeclaPulgar_D;
    public GameObject TeclaIndice_D;
    public GameObject TeclaCorazon_D;
    public GameObject TeclaAnular_D;
    public GameObject TeclaMeñique_D;
    public GameObject TeclaPulgar_I;
    public GameObject TeclaIndice_I;
    public GameObject TeclaCorazon_I;
    public GameObject TeclaAnular_I;
    public GameObject TeclaMeñique_I;
    public Material[] materials;
    public Renderer rendTeclaPD;
    public Renderer rendTeclaID;
    public Renderer rendTeclaCD;
    public Renderer rendTeclaAD;
    public Renderer rendTeclaMD;
    public Renderer rendTeclaPI;
    public Renderer rendTeclaII;
    public Renderer rendTeclaCI;
    public Renderer rendTeclaAI;
    public Renderer rendTeclaMI;
    public ParticleSystem pS_PD;
    public ParticleSystem pS_ID;
    public ParticleSystem pS_CD;
    public ParticleSystem pS_AD;
    public ParticleSystem pS_MD;
    public ParticleSystem pS_PI;
    public ParticleSystem pS_II;
    public ParticleSystem pS_CI;
    public ParticleSystem pS_AI;
    public ParticleSystem pS_MI;
    public Canvas End;
    public static int aux;
    public static int enableSound_MI = 0;
    public static int enableSound_AI = 0;
    public static int enableSound_CI = 0;
    public static int enableSound_II = 0;
    public static int enableSound_PI = 0;
    public static int enableSound_MD = 0;
    public static int enableSound_AD = 0;
    public static int enableSound_CD = 0;
    public static int enableSound_ID = 0;
    public static int enableSound_PD = 0;
    private int index = 1;
    public static float[] tiempos1mano_PD = new float[4];
    public static float[] tiempos1mano_ID = new float[4];
    public static float[] tiempos1mano_CD = new float[4];
    public static float[] tiempos1mano_AD = new float[4];
    public static float[] tiempos1mano_MD = new float[4];
```

```
public static float[] tiempos1mano_PI = new float[4];
public static float[] tiempos1mano_II = new float[4];
public static float[] tiempos1mano_CI = new float[4];
public static float[] tiempos1mano_AI = new float[4];
public static float[] tiempos1mano_MI = new float[4];
public static float[] tiempos2manos_PD = new float[4];
public static float[] tiempos2manos_ID = new float[4];
public static float[] tiempos2manos_CD = new float[4];
public static float[] tiempos2manos_AD = new float[4];
public static float[] tiempos2manos_MD = new float[4];
public static float[] tiempos2manos_PI = new float[4];
public static float[] tiempos2manos_II = new float[4];
public static float[] tiempos2manos_CI = new float[4];
public static float[] tiempos2manos_AI = new float[4];
public static float[] tiempos2manos_MI = new float[4];

void Start()  {
   End = End.GetComponent<Canvas>();
   End.enabled = false;
   ParticleSystem.EmissionModule em0 = pS_MI.emission;
   ParticleSystem.EmissionModule em1 = pS_AI.emission;
   ParticleSystem.EmissionModule em2 = pS_CI.emission;
   ParticleSystem.EmissionModule em3 = pS_II.emission;
   ParticleSystem.EmissionModule em4 = pS_PI.emission;
   ParticleSystem.EmissionModule em5 = pS_PD.emission;
   ParticleSystem.EmissionModule em6 = pS_ID.emission;
   ParticleSystem.EmissionModule em7 = pS_CD.emission;
   ParticleSystem.EmissionModule em8 = pS_AD.emission;
   ParticleSystem.EmissionModule em9 = pS_MD.emission;
   em0.enabled = false;
   em1.enabled = false;
   em2.enabled = false;
   em3.enabled = false;
   em4.enabled = false;
   em5.enabled = false;
   em6.enabled = false;
   em7.enabled = false;
   em8.enabled = false;
   em9.enabled = false;   }

public void ComenzarColores()  {
   StartCoroutine(CheckBool());   }

IEnumerator CheckBool()  {
   ParticleSystem.EmissionModule em0 = pS_MI.emission;
   ParticleSystem.EmissionModule em1 = pS_AI.emission;
   ParticleSystem.EmissionModule em2 = pS_CI.emission;
   ParticleSystem.EmissionModule em3 = pS_II.emission;
   ParticleSystem.EmissionModule em4 = pS_PI.emission;
   ParticleSystem.EmissionModule em5 = pS_PD.emission;
   ParticleSystem.EmissionModule em6 = pS_ID.emission;
   ParticleSystem.EmissionModule em7 = pS_CD.emission;
   ParticleSystem.EmissionModule em8 = pS_AD.emission;
   ParticleSystem.EmissionModule em9 = pS_MD.emission;
   int cont1 = 0;
   int cont2 = 0;
   aux = 0;
   int random1 = 0;
   int random2 = 0;
   int random3 = 0;
   int random4 = 0;
   int random5 = 0;
   int random6 = 0;
   int random7 = 0;
   int random8 = 0;
   int random9 = 0;
   int random10 = 0;
   System.Random rdn = new System.Random();
   Stopwatch stopWatch = new Stopwatch();
   System.TimeSpan timeSpan = new System.TimeSpan();
   float segundos;
```

```
float mili;
float suma;

//AMBAS MANOS
if (Application.loadedLevel == 4)     {
   for (int i = 0; i < 4; i++)          {
      tiempos2manos_PD[i] = 0;
      tiempos2manos_ID[i] = 0;
      tiempos2manos_CD[i] = 0;
      tiempos2manos_AD[i] = 0;
      tiempos2manos_MD[i] = 0;
      tiempos2manos_PI[i] = 0;
      tiempos2manos_II[i] = 0;
      tiempos2manos_CI[i] = 0;
      tiempos2manos_AI[i] = 0;
      tiempos2manos_MI[i] = 0;         }
   //EN ORDEN AMBAS MANOS
   while (cont1 < 2)          {
      switch (aux)             {
         case 0:
            em0.enabled = true;
            enableSound_MI = 1;
            rendTeclaMI = TeclaMeñique_I.GetComponent<Renderer>();
            rendTeclaMI.enabled = true;
            index += 1;
            if (index == materials.Length + 1)       {
               index = 1;            }
            rendTeclaMI.sharedMaterial = materials[index - 1];
            stopWatch.Start();
            yield return new WaitUntil(() => aux == 1);
            stopWatch.Stop();
            timeSpan = stopWatch.Elapsed;
            segundos = timeSpan.Seconds;
            mili = timeSpan.Milliseconds;
            suma = segundos + mili / 1000;
            if (cont1 == 0)              {
               tiempos2manos_MI[0] = suma;           }
            if (cont1 == 1)                 {
               tiempos2manos_MI[1] = suma;                  }
            stopWatch.Reset();
            index = 1;
            enableSound_MI = 0;
            em0.enabled = false;
            break;
         case 1:
            em1.enabled = true;
            enableSound_AI = 1;
            rendTeclaAI = TeclaAnular_I.GetComponent<Renderer>();
            rendTeclaAI.enabled = true;
            index += 1;
            if (index == materials.Length + 1)        {
               index = 1;             }
            rendTeclaAI.sharedMaterial = materials[index - 1];
            stopWatch.Start();
            yield return new WaitUntil(() => aux == 2);
            stopWatch.Stop();
            timeSpan = stopWatch.Elapsed;
            segundos = timeSpan.Seconds;
            mili = timeSpan.Milliseconds;
            suma = segundos + mili / 1000;
            if (cont1 == 0)            {
               tiempos2manos_AI[0] = suma;              }
            if (cont1 == 1)              {
               tiempos2manos_AI[1] = suma;    }
            stopWatch.Reset();
            index = 1;
            enableSound_AI = 0;
            em1.enabled = false;
            break;
         case 2:
            em2.enabled = true;
```

```
    enableSound_CI = 1;
    rendTeclaCI = TeclaCorazon_I.GetComponent<Renderer>();
    rendTeclaCI.enabled = true;
    index += 1;
    if (index == materials.Length + 1)          {
      index = 1;              }
    rendTeclaCI.sharedMaterial = materials[index - 1];
    stopWatch.Start();
    yield return new WaitUntil(() => aux == 3);
    stopWatch.Stop();
    timeSpan = stopWatch.Elapsed;
    segundos = timeSpan.Seconds;
    mili = timeSpan.Milliseconds;
    suma = segundos + mili / 1000;
    if (cont1 == 0)             {
      tiempos2manos_CI[0] = suma;             }
    if (cont1 == 1)              {
      tiempos2manos_CI[1] = suma;             }
    stopWatch.Reset();
    index = 1;
    enableSound_CI = 0;
    em2.enabled = false;
    break;
  case 3:
    em3.enabled = true;
    enableSound_II = 1;
    rendTeclaII = TeclaIndice_I.GetComponent<Renderer>();
    rendTeclaII.enabled = true;
    index += 1;
    if (index == materials.Length + 1)          {
      index = 1;              }
    rendTeclaII.sharedMaterial = materials[index - 1];
    stopWatch.Start();
    yield return new WaitUntil(() => aux == 4);
    stopWatch.Stop();
    timeSpan = stopWatch.Elapsed;
    segundos = timeSpan.Seconds;
    mili = timeSpan.Milliseconds;
    suma = segundos + mili / 1000;
    if (cont1 == 0)          {
      tiempos2manos_II[0] = suma;         }
    if (cont1 == 1)             {
      tiempos2manos_II[1] = suma;         }
    stopWatch.Reset();
    index = 1;
    enableSound_II = 0;
    em3.enabled = false;
    break;
  case 4:
    em4.enabled = true;
    enableSound_PI = 1;
    rendTeclaPI = TeclaPulgar_I.GetComponent<Renderer>();
    rendTeclaPI.enabled = true;
    index += 1;
    if (index == materials.Length + 1)                  {
      index = 1;                  }
    rendTeclaPI.sharedMaterial = materials[index - 1];
    stopWatch.Start();
    yield return new WaitUntil(() => aux == 5);
    stopWatch.Stop();
    timeSpan = stopWatch.Elapsed;
    segundos = timeSpan.Seconds;
    mili = timeSpan.Milliseconds;
    suma = segundos + mili / 1000;
    if (cont1 == 0)                  {
      tiempos2manos_PI[0] = suma;                 }
    if (cont1 == 1)                  {
      tiempos2manos_PI[1] = suma;                 }
    stopWatch.Reset();
    index = 1;
    enableSound_PI = 0;
```

```
    em4.enabled = false;
    break;
  case 5:
    em5.enabled = true;
    enableSound_PD = 1;
    rendTeclaPD = TeclaPulgar_D.GetComponent<Renderer>();
    rendTeclaPD.enabled = true;
    index += 1;
    if (index == materials.Length + 1)                    {
      index = 1;                    }
    rendTeclaPD.sharedMaterial = materials[index - 1];
    stopWatch.Start();
    yield return new WaitUntil(() => aux == 6);
    stopWatch.Stop();
    timeSpan = stopWatch.Elapsed;
    segundos = timeSpan.Seconds;
    mili = timeSpan.Milliseconds;
    suma = segundos + mili / 1000;
    if (cont1 == 0)                    {
      tiempos2manos_PD[0] = suma;                    }
    if (cont1 == 1)                    {
      tiempos2manos_PD[1] = suma;                    }
    stopWatch.Reset();
    index = 1;
    enableSound_PD = 0;
    em5.enabled = false;
    break;
  case 6:
    em6.enabled = true;
    enableSound_ID = 1;
    rendTeclaID = TeclaIndice_D.GetComponent<Renderer>();
    rendTeclaID.enabled = true;
    index += 1;
    if (index == materials.Length + 1)                    {
      index = 1;                    }
    rendTeclaID.sharedMaterial = materials[index - 1];
    stopWatch.Start();
    yield return new WaitUntil(() => aux == 7);
    stopWatch.Stop();
    timeSpan = stopWatch.Elapsed;
    segundos = timeSpan.Seconds;
    mili = timeSpan.Milliseconds;
    suma = segundos + mili / 1000;
    if (cont1 == 0)                    {
      tiempos2manos_ID[0] = suma;                    }
    if (cont1 == 1)                    {
      tiempos2manos_ID[1] = suma;                    }
    stopWatch.Reset();
    index = 1;
    enableSound_ID = 0;
    em6.enabled = false;
    break;
  case 7:
    em7.enabled = true;
    enableSound_CD = 1;
    rendTeclaCD = TeclaCorazon_D.GetComponent<Renderer>();
    rendTeclaCD.enabled = true;
    index += 1;
    if (index == materials.Length + 1)                    {
      index = 1;                    }
    rendTeclaCD.sharedMaterial = materials[index - 1];
    stopWatch.Start();
    yield return new WaitUntil(() => aux == 8);
    stopWatch.Stop();
    timeSpan = stopWatch.Elapsed;
    segundos = timeSpan.Seconds;
    mili = timeSpan.Milliseconds;
    suma = segundos + mili / 1000;
    if (cont1 == 0)                    {
      tiempos2manos_CD[0] = suma;                    }
    if (cont1 == 1)                    {
```

```
            tiempos2manos_CD[1] = suma;                    }
        stopWatch.Reset();
        index = 1;
        enableSound_CD = 0;
        em7.enabled = false;
        break;
      case 8:
        em8.enabled = true;
        enableSound_AD = 1;
        rendTeclaAD = TeclaAnular_D.GetComponent<Renderer>();
        rendTeclaAD.enabled = true;
        index += 1;
        if (index == materials.Length + 1)                  {
          index = 1;                  }
        rendTeclaAD.sharedMaterial = materials[index - 1];
        stopWatch.Start();
        yield return new WaitUntil(() => aux == 9);
        stopWatch.Stop();
        timeSpan = stopWatch.Elapsed;
        segundos = timeSpan.Seconds;
        mili = timeSpan.Milliseconds;
        suma = segundos + mili / 1000;
        if (cont1 == 0)                  {
          tiempos2manos_AD[0] = suma;                  }
        if (cont1 == 1)                  {
          tiempos2manos_AD[1] = suma;                  }
        stopWatch.Reset();
        index = 1;
        enableSound_AD = 0;
        em8.enabled = false;
        break;
      case 9:
        em9.enabled = true;
        enableSound_MD = 1;
        rendTeclaMD = TeclaMeñique_D.GetComponent<Renderer>();
        rendTeclaMD.enabled = true;
        index += 1;
        if (index == materials.Length + 1)                  {
          index = 1;                  }
        rendTeclaMD.sharedMaterial = materials[index - 1];
        stopWatch.Start();
        yield return new WaitUntil(() => aux == 10);
        stopWatch.Stop();
        timeSpan = stopWatch.Elapsed;
        segundos = timeSpan.Seconds;
        mili = timeSpan.Milliseconds;
        suma = segundos + mili / 1000;
        if (cont1 == 0)                  {
          tiempos2manos_MD[0] = suma;                  }
        if (cont1 == 1)                  {
          tiempos2manos_MD[1] = suma;                  }
        stopWatch.Reset();
        index = 1;
        enableSound_MD = 0;
        aux = 0;
        cont1 += 1;
        if (cont1 == 2)                  {
          cont2 = 1;                  }
        em9.enabled = false;
        break;            }
}
//ALEATORIO AMBAS MANOS
if (cont2 == 1)          {
  while (cont2 < 3)             {
    int n = 0;
    int value = 0;
    while (n < 10)               {
      switch (n) {
        case 0:
          random1 = rdn.Next(1, 11);
          value = random1;
```

```
    break;
case 1:
    random2 = rdn.Next(1, 11);
    if (random2 == random1)                    {
        while (random2 == random1)                    {
            random2 = rdn.Next(1, 11);                    }
    }
    value = random2;
    break;
case 2:
    random3 = rdn.Next(1, 11);
    if ((random3 == random1) || (random3 == random2))                    {
        while ((random3 == random1) || (random3 == random2))                    {
            random3 = rdn.Next(1, 11);                    }
    }
    value = random3;
    break;
case 3:
    random4 = rdn.Next(1, 11);
    if ((random4 == random1) || (random4 == random2) || (random4 == random3))                    {
        while ((random4 == random1) || (random4 == random2) || (random4 == random3))                    {
            random4 = rdn.Next(1, 11);                    }
    }
    value = random4;
    break;
case 4:
    random5 = rdn.Next(1, 11);
    if ((random5 == random1) || (random5 == random2) || (random5 == random3) || (random5 == random4))    {
        while ((random5==random1) || (random5 == random2) || (random5 == random3) || (random5 == random4)){
            random5 = rdn.Next(1, 11);                    }
    }
    value = random5;
    break;
case 5:
    random6 = rdn.Next(1, 11);
    if ((random6 == random1) || (random6 == random2) || (random6 == random3) || (random6 == random4) ||
        (random6 == random5))                    {
        while ((random6 == random1) || (random6 == random2) || (random6 == random3) || (random6 == random4)
            || (random6 == random5))                    {
            random6 = rdn.Next(1, 11);                    }
    }
    value = random6;
    break;
case 6:
    random7 = rdn.Next(1, 11);
    if ((random7 == random1) || (random7 == random2) || (random7 == random3) || (random7 == random4) ||
        (random7 == random5) || (random7 == random6))                    {
        while ((random7 == random1) || (random7 == random2) || (random7 == random3) || (random7 == random4)
            || (random7 == random5) || (random7 == random6))                    {
            random7 = rdn.Next(1, 11);                    }
    }
    value = random7;
    break;
case 7:
    random8 = rdn.Next(1, 11);
    if ((random8 == random1) || (random8 == random2) || (random8 == random3) || (random8 == random4) ||
        (random8 == random5) || (random8 == random6) || (random8 == random7))                    {
        while ((random8 == random1) || (random8 == random2) || (random8 == random3) || (random8 == random4)
            || (random8 == random5) || (random8 == random6) || (random8 == random7))                    {
            random8 = rdn.Next(1, 11);                    }
    }
    value = random8;
    break;
case 8:
    random9 = rdn.Next(1, 11);
    if ((random9 == random1) || (random9 == random2) || (random9 == random3) || (random9 == random4) ||
        (random9 == random5) || (random9 == random6) || (random9 == random7) || (random9 == random8))    {
        while ((random9 == random1) || (random9 == random2) || (random9 == random3) || (random9 == random4)
            || (random9 == random5) || (random9 == random6) || (random9 == random7) || (random9 == random8)) {
            random9 = rdn.Next(1, 11);                    }
    }
```

```
            value = random9;
            break;
        case 9:
            random10 = rdn.Next(1, 11);
            if ((random10 == random1) || (random10 == random2) || (random10 == random3) || (random10 == random4)
                || (random10 == random5) || (random10 == random6) || (random10 == random7) || (random10 ==
                random8) || (random10 == random9))                {
              while ((random10 == random1) || (random10 == random2) || (random10 == random3) || (random10 ==
                random4) || (random10 == random5) || (random10 == random6) || (random10 == random7) || (random10
                == random8) || (random10 == random9))                    {
                 random10 = rdn.Next(1, 11);                        }
            }
            value = random10;
            break;                }
        switch (value)                {
          case 1:
            em0.enabled = true;
            enableSound_MI = 1;
            rendTeclaMI = TeclaMeñique_I.GetComponent<Renderer>();
            rendTeclaMI.enabled = true;
            index += 1;
            if (index == materials.Length + 1)                        {
              index = 1;                        }
            rendTeclaMI.sharedMaterial = materials[index - 1];
            stopWatch.Start();
            yield return new WaitUntil(() => aux == 1);
            stopWatch.Stop();
            timeSpan = stopWatch.Elapsed;
            segundos = timeSpan.Seconds;
            mili = timeSpan.Milliseconds;
            suma = segundos + mili / 1000;
            if (cont2 == 1)                        {
              tiempos2manos_MI[2] = suma;                        }
            if (cont2 == 2)                        {
              tiempos2manos_MI[3] = suma;                        }
            stopWatch.Reset();
            enableSound_MI = 0;
            em0.enabled = false;
            index = 1;
            break;
          case 2:
            em1.enabled = true;
            enableSound_AI = 1;
            rendTeclaAI = TeclaAnular_I.GetComponent<Renderer>();
            rendTeclaAI.enabled = true;
            index += 1;
            if (index == materials.Length + 1)                        {
              index = 1;                        }
            rendTeclaAI.sharedMaterial = materials[index - 1];
            stopWatch.Start();
            yield return new WaitUntil(() => aux == 2);
            stopWatch.Stop();
            timeSpan = stopWatch.Elapsed;
            segundos = timeSpan.Seconds;
            mili = timeSpan.Milliseconds;
            suma = segundos + mili / 1000;
            if (cont2 == 1)                        {
              tiempos2manos_AI[2] = suma;                        }
            if (cont2 == 2)                        {
              tiempos2manos_AI[3] = suma;                        }
            stopWatch.Reset();
            enableSound_AI = 0;
            em1.enabled = false;
            index = 1;
            break;
          case 3:
            em2.enabled = true;
            enableSound_CI = 1;
            rendTeclaCI = TeclaCorazon_I.GetComponent<Renderer>();
            rendTeclaCI.enabled = true;
            index += 1;
```

```
    if (index == materials.Length + 1)                    {
      index = 1;                    }
    rendTeclaCI.sharedMaterial = materials[index - 1];
    stopWatch.Start();
    yield return new WaitUntil(() => aux == 3);
    stopWatch.Stop();
    timeSpan = stopWatch.Elapsed;
    segundos = timeSpan.Seconds;
    mili = timeSpan.Milliseconds;
    suma = segundos + mili / 1000;
    if (cont2 == 1)                    {
      tiempos2manos_CI[2] = suma;                    }
    if (cont2 == 2)                    {
      tiempos2manos_CI[3] = suma;                    }
    stopWatch.Reset();
    enableSound_CI = 0;
    em2.enabled = false;
    index = 1;
    break;
  case 4:
    em3.enabled = true;
    enableSound_II = 1;
    rendTeclaII = TeclaIndice_I.GetComponent<Renderer>();
    rendTeclaII.enabled = true;
    index += 1;
    if (index == materials.Length + 1)                    {
      index = 1;                    }
    rendTeclaII.sharedMaterial = materials[index - 1];
    stopWatch.Start();
    yield return new WaitUntil(() => aux == 4);
    stopWatch.Stop();
    timeSpan = stopWatch.Elapsed;
    segundos = timeSpan.Seconds;
    mili = timeSpan.Milliseconds;
    suma = segundos + mili / 1000;
    if (cont2 == 1)                    {
      tiempos2manos_II[2] = suma;                    }
    if (cont2 == 2)                    {
      tiempos2manos_II[3] = suma;                    }
    stopWatch.Reset();
    enableSound_II = 0;
    em3.enabled = false;
    index = 1;
    break;
  case 5:
    em4.enabled = true;
    enableSound_PI = 1;
    rendTeclaPI = TeclaPulgar_I.GetComponent<Renderer>();
    rendTeclaPI.enabled = true;
    index += 1;
    if (index == materials.Length + 1)                    {
      index = 1;                    }
    rendTeclaPI.sharedMaterial = materials[index - 1];
    stopWatch.Start();
    yield return new WaitUntil(() => aux == 5);
    stopWatch.Stop();
    timeSpan = stopWatch.Elapsed;
    segundos = timeSpan.Seconds;
    mili = timeSpan.Milliseconds;
    suma = segundos + mili / 1000;
    if (cont2 == 1)                    {
      tiempos2manos_PI[2] = suma;                    }
    if (cont2 == 2)                    {
      tiempos2manos_PI[3] = suma;                    }
    stopWatch.Reset();
    enableSound_PI = 0;
    em4.enabled = false;
    index = 1;
    break;
  case 6:
    em5.enabled = true;
```

```
    enableSound_PD = 1;
    rendTeclaPD = TeclaPulgar_D.GetComponent<Renderer>();
    rendTeclaPD.enabled = true;
    index += 1;
    if (index == materials.Length + 1)                    {
      index = 1;                    }
    rendTeclaPD.sharedMaterial = materials[index - 1];
    stopWatch.Start();
    yield return new WaitUntil(() => aux == 6);
    stopWatch.Stop();
    timeSpan = stopWatch.Elapsed;
    segundos = timeSpan.Seconds;
    mili = timeSpan.Milliseconds;
    suma = segundos + mili / 1000;
    if (cont2 == 1)                    {
      tiempos2manos_PD[2] = suma;                    }
    if (cont2 == 2)                    {
      tiempos2manos_PD[3] = suma;                    }
    stopWatch.Reset();
    enableSound_PD = 0;
    em5.enabled = false;
    index = 1;
    break;
  case 7:
    em6.enabled = true;
    enableSound_ID = 1;
    rendTeclaID = TeclaIndice_D.GetComponent<Renderer>();
    rendTeclaID.enabled = true;
    index += 1;
    if (index == materials.Length + 1)                    {
      index = 1;                    }
    rendTeclaID.sharedMaterial = materials[index - 1];
    stopWatch.Start();
    yield return new WaitUntil(() => aux == 7);
    stopWatch.Stop();
    timeSpan = stopWatch.Elapsed;
    segundos = timeSpan.Seconds;
    mili = timeSpan.Milliseconds;
    suma = segundos + mili / 1000;
    if (cont2 == 1)                    {
      tiempos2manos_ID[2] = suma;                    }
    if (cont2 == 2)                    {
      tiempos2manos_ID[3] = suma;                    }
    stopWatch.Reset();
    enableSound_ID = 0;
    em6.enabled = false;
    index = 1;
    break;
  case 8:
    em7.enabled = true;
    enableSound_CD = 1;
    rendTeclaCD = TeclaCorazon_D.GetComponent<Renderer>();
    rendTeclaCD.enabled = true;
    index += 1;
    if (index == materials.Length + 1)                    {
      index = 1;                    }
    rendTeclaCD.sharedMaterial = materials[index - 1];
    stopWatch.Start();
    yield return new WaitUntil(() => aux == 8);
    stopWatch.Stop();
    timeSpan = stopWatch.Elapsed;
    segundos = timeSpan.Seconds;
    mili = timeSpan.Milliseconds;
    suma = segundos + mili / 1000;
    if (cont2 == 1)                    {
      tiempos2manos_CD[2] = suma;                    }
    if (cont2 == 2)                    {
      tiempos2manos_CD[3] = suma;                    }
    stopWatch.Reset();
    enableSound_CD = 0;
    em7.enabled = false;
```

```
                index = 1;
                break;
            case 9:
                em8.enabled = true;
                enableSound_AD = 1;
                rendTeclaAD = TeclaAnular_D.GetComponent<Renderer>();
                rendTeclaAD.enabled = true;
                index += 1;
                if (index == materials.Length + 1)                    {
                    index = 1;                    }
                rendTeclaAD.sharedMaterial = materials[index - 1];
                stopWatch.Start();
                yield return new WaitUntil(() => aux == 9);
                stopWatch.Stop();
                timeSpan = stopWatch.Elapsed;
                segundos = timeSpan.Seconds;
                mili = timeSpan.Milliseconds;
                suma = segundos + mili / 1000;
                if (cont2 == 1)                    {
                    tiempos2manos_AD[2] = suma;                    }
                if (cont2 == 2)                    {
                    tiempos2manos_AD[3] = suma;                    }
                stopWatch.Reset();
                enableSound_AD = 0;
                em8.enabled = false;
                index = 1;
                break;
            case 10:
                em9.enabled = true;
                enableSound_MD = 1;
                rendTeclaMD = TeclaMeñique_D.GetComponent<Renderer>();
                rendTeclaMD.enabled = true;
                index += 1;
                if (index == materials.Length + 1)                    {
                    index = 1;                    }
                rendTeclaMD.sharedMaterial = materials[index - 1];
                stopWatch.Start();
                yield return new WaitUntil(() => aux == 10);
                stopWatch.Stop();
                timeSpan = stopWatch.Elapsed;
                segundos = timeSpan.Seconds;
                mili = timeSpan.Milliseconds;
                suma = segundos + mili / 1000;
                if (cont2 == 1)                    {
                    tiempos2manos_MD[2] = suma;                    }
                if (cont2 == 2)                    {
                    tiempos2manos_MD[3] = suma;                    }
                stopWatch.Reset();
                enableSound_MD = 0;
                em9.enabled = false;
                index = 1;
                break;                    }
        n += 1;        }
      cont2 += 1;          }
  }
  End.enabled = true;   }
}
```

IgnoreCol

Debido a que este script tiene muchas líneas de código, en este anexo se incluye tan solo la parte que involucra ignorar la colisión entre la tecla correspondiente al dedo pulgar de la mano derecha con los dedos que no son el pulgar. Para el resto de teclas se realizará lo mismo ignorando los dedos que no conciernen a cada tecla.

```
using UnityEngine;
using System.Collections;

public class IgnoreCol : MonoBehaviour {
```

```
    public GameObject floor;
    public GameObject TPD;
    public GameObject TID;
    public GameObject TCD;
    public GameObject TAD;
    public GameObject TMD;
    public GameObject TPI;
    public GameObject TII;
    public GameObject TCI;
    public GameObject TAI;
    public GameObject TMI;
    public GameObject Pulgar_bone1_D;
    public GameObject Pulgar_bone2_D;
    public GameObject Pulgar_bone3_D;
    public GameObject Indice_bone1_D;
    public GameObject Indice_bone2_D;
    public GameObject Indice_bone3_D;
    public GameObject Corazon_bone1_D;
    public GameObject Corazon_bone2_D;
    public GameObject Corazon_bone3_D;
    public GameObject Anular_bone1_D;
    public GameObject Anular_bone2_D;
    public GameObject Anular_bone3_D;
    public GameObject Meñique_bone1_D;
    public GameObject Meñique_bone2_D;
    public GameObject Meñique_bone3_D;
    public GameObject Pulgar_bone1_I;
    public GameObject Pulgar_bone2_I;
    public GameObject Pulgar_bone3_I;
    public GameObject Indice_bone1_I;
    public GameObject Indice_bone2_I;
    public GameObject Indice_bone3_I;
    public GameObject Corazon_bone1_I;
    public GameObject Corazon_bone2_I;
    public GameObject Corazon_bone3_I;
    public GameObject Anular_bone1_I;
    public GameObject Anular_bone2_I;
    public GameObject Anular_bone3_I;
    public GameObject Meñique_bone1_I;
    public GameObject Meñique_bone2_I;
    public GameObject Meñique_bone3_I;

void Update () {
      //Tecla pulgar derecha con los dedos de la mano derecha

      Physics.IgnoreCollision(TPD.GetComponent<Collider>(), Indice_bone1_D.GetComponent<Collider>(), true);
      Physics.IgnoreCollision(TPD.GetComponent<Collider>(), Indice_bone2_D.GetComponent<Collider>(), true);
      Physics.IgnoreCollision(TPD.GetComponent<Collider>(), Indice_bone3_D.GetComponent<Collider>(), true);
      Physics.IgnoreCollision(TPD.GetComponent<Collider>(), Corazon_bone1_D.GetComponent<Collider>(), true);
      Physics.IgnoreCollision(TPD.GetComponent<Collider>(), Corazon_bone2_D.GetComponent<Collider>(), true);
      Physics.IgnoreCollision(TPD.GetComponent<Collider>(), Corazon_bone3_D.GetComponent<Collider>(), true);
      Physics.IgnoreCollision(TPD.GetComponent<Collider>(), Anular_bone1_D.GetComponent<Collider>(), true);
      Physics.IgnoreCollision(TPD.GetComponent<Collider>(), Anular_bone2_D.GetComponent<Collider>(), true);
      Physics.IgnoreCollision(TPD.GetComponent<Collider>(), Anular_bone3_D.GetComponent<Collider>(), true);
      Physics.IgnoreCollision(TPD.GetComponent<Collider>(), Meñique_bone1_D.GetComponent<Collider>(), true);
      Physics.IgnoreCollision(TPD.GetComponent<Collider>(), Meñique_bone2_D.GetComponent<Collider>(), true);
      Physics.IgnoreCollision(TPD.GetComponent<Collider>(), Meñique_bone3_D.GetComponent<Collider>(), true);
}
```

Tecla_X

Existe un script con este nombre para cada tecla, siendo X el nombre del dedo que le atañe. En este anexo tan solo se incluye el código perteneciente a Tecla_MeñiquеI (correspondiente al meñique izquierdo), siendo el resto de scripts semejantes a este pero variando los dedos, teclas y valor de la variable auxiliar (entre 1 y 10).

```
using UnityEngine;
using System.Collections;
```

```
public class Tecla_MeñiqueI : MonoBehaviour {
   public Material[] materials;
   public Renderer rend;
   public float semitone_offset = 0;
   private int index = 1;

   void Start()   {
      rend = GetComponent<Renderer>();
      rend.enabled = true;   }

   public void OnCollisionEnter(Collision collision)   {
      if (collision.gameObject.name == "floor")      {
         if (Colores.enableSound_MI == 1)         {
            AudioSource audio = GetComponent<AudioSource>();
            audio.pitch = Mathf.Pow(2f, semitone_offset / 12.0f);
            audio.Play();          }
         index += 1;
         if (index == materials.Length + 1)         {
            index = 1;          }
         rend.sharedMaterial = materials[index - 1];
         Colores.aux = 1;      }
   }
}
```

Botones

Debido a la extensión del código se ha omitido parte del código donde se programa cómo se guardan los datos de los ejercicios en el archivo. Así se muestra en la función "NextLevel()" cómo se guardan los correspondientes al ejercicio de la mano derecha, en "PreviousLevel()" los de la mano izquierda y en "ExitGame()" los del ejercicio que involucra ambas manos. Estas tres partes del código deberían aparecer todas en las tres funciones mencionadas y no uno en cada una.

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
using System.IO;

public class Botones : MonoBehaviour {
   public Canvas quitMenu;
   public Button next;
   public Button previous;
   public Button exit;
   public Button conf;
   public Canvas Instrucciones;
   public Button play;

   void Start () {
      quitMenu = quitMenu.GetComponent<Canvas>();
      quitMenu.enabled = false;
      next = next.GetComponent<Button>();
      exit = exit.GetComponent<Button>();
      conf = conf.GetComponent<Button>();
      Instrucciones = Instrucciones.GetComponent<Canvas>();
      play = play.GetComponent<Button>();
      Instrucciones.enabled = true;   }

   public void PlayPress()   {
      Instrucciones.enabled = false;
      next.enabled = true;   }

   public void ExitPress()   {
      quitMenu.enabled = true;
      next.enabled = false;
      exit.enabled = false;
      conf.enabled = false;   }

   public void NoPress()   {
      quitMenu.enabled = false;
      next.enabled = true;
```

```
    exit.enabled = true;
    conf.enabled = true;    }

  public void NextLevel()    {
    if ((Application.loadedLevel == 2) && (Colores.tiempos1mano_PD[0] != 0)&& (File.Exists(GetInput.ruta)))       {
      using (StreamWriter writer = File.AppendText(GetInput.ruta))          {
        writer.WriteLine(" ");
        writer.WriteLine("TIEMPOS DURANTE LA PRUEBA CON LA MANO DERECHA");
        writer.WriteLine("Pulgar derecho: ," + Colores.tiempos1mano_PD[0] + ", " + Colores.tiempos1mano_PD[1] + ", " +
          Colores.tiempos1mano_PD[2] + ", " + Colores.tiempos1mano_PD[3]);
        writer.WriteLine("Indice derecho: ," + Colores.tiempos1mano_ID[0] + ", " + Colores.tiempos1mano_ID[1] + ", " +
          Colores.tiempos1mano_ID[2] + ", " + Colores.tiempos1mano_ID[3]);
        writer.WriteLine("Corazon derecho: ," + Colores.tiempos1mano_CD[0] + ", " + Colores.tiempos1mano_CD[1] + ", " +
          Colores.tiempos1mano_CD[2] + ", " + Colores.tiempos1mano_CD[3]);
        writer.WriteLine("Anular derecho: ," + Colores.tiempos1mano_AD[0] + ", " + Colores.tiempos1mano_AD[1] + ", " +
          Colores.tiempos1mano_AD[2] + ", " + Colores.tiempos1mano_AD[3]);
        writer.WriteLine("Meñique derecho: ," + Colores.tiempos1mano_MD[0] + ", " + Colores.tiempos1mano_MD[1] + ", " +
          Colores.tiempos1mano_MD[2] + ", " + Colores.tiempos1mano_MD[3]);          }
    }
    int NextLevel;
    NextLevel = Application.loadedLevel + 1;
    Application.LoadLevel(NextLevel);    }

  public void PreviousLevel()    {
    if ((Application.loadedLevel == 3) && (Colores.tiempos1mano_MI[0] != 0) && (File.Exists(GetInput.ruta)))       {
      using (StreamWriter writer = File.AppendText(GetInput.ruta))          {
        writer.WriteLine(" ");
        writer.WriteLine("TIEMPOS DURANTE LA PRUEBA CON LA MANO IZQUIERDA");
        writer.WriteLine("Pulgar izquierdo: ," + Colores.tiempos1mano_PI[0] + ", " + Colores.tiempos1mano_PI[1] + ", " +
          Colores.tiempos1mano_PI[2] + ", " + Colores.tiempos1mano_PD[3]);
        writer.WriteLine("Indice izquierdo: ," + Colores.tiempos1mano_II[0] + ", " + Colores.tiempos1mano_II[1] + ", " +
          Colores.tiempos1mano_II[2] + ", " + Colores.tiempos1mano_ID[3]);
        writer.WriteLine("Corazon izquierdo: ," + Colores.tiempos1mano_CI[0] + ", " + Colores.tiempos1mano_CI[1] + ", " +
          Colores.tiempos1mano_CI[2] + ", " + Colores.tiempos1mano_CI[3]);
        writer.WriteLine("Anular izquierdo: ," + Colores.tiempos1mano_AI[0] + ", " + Colores.tiempos1mano_AI[1] + ", " +
          Colores.tiempos1mano_AI[2] + ", " + Colores.tiempos1mano_AI[3]);
        writer.WriteLine("Meñique izquierdo: ," + Colores.tiempos1mano_MI[0] + ", " + Colores.tiempos1mano_MI[1] + ", " +
          Colores.tiempos1mano_MI[2] + ", " + Colores.tiempos1mano_MI[3]);          }
    }
    int PrevLevel;
    PrevLevel = Application.loadedLevel - 1;
    Application.LoadLevel(PrevLevel);    }

  public void Conf()    {
    Sliders.level = Application.loadedLevel;
    Application.LoadLevel(1);    }

  public void ExitGame()    {
    if ((Application.loadedLevel == 4) && (Colores.tiempos2manos_MI[0] != 0) && (File.Exists(GetInput.ruta)))       {
      using (StreamWriter writer = File.AppendText(GetInput.ruta))          {
        writer.WriteLine(" ");
        writer.WriteLine("TIEMPOS DE LA MANO IZQUIERDA DURANTE LA PRUEBA CON AMBAS MANOS");
        writer.WriteLine("Pulgar izquierdo: ," + Colores.tiempos2manos_PI[0] + ", " + Colores.tiempos2manos_PI[1] + ", " +
          Colores.tiempos2manos_PI[2] + ", " + Colores.tiempos2manos_PD[3]);
        writer.WriteLine("Indice izquierdo: ," + Colores.tiempos2manos_II[0] + ", " + Colores.tiempos2manos_II[1] + ", " +
          Colores.tiempos2manos_II[2] + ", " + Colores.tiempos2manos_ID[3]);
        writer.WriteLine("Corazon izquierdo: ," + Colores.tiempos2manos_CI[0] + ", " + Colores.tiempos2manos_CI[1] + ", " +
          Colores.tiempos2manos_CI[2] + ", " + Colores.tiempos2manos_CI[3]);
        writer.WriteLine("Anular izquierdo: ," + Colores.tiempos2manos_AI[0] + ", " + Colores.tiempos2manos_AI[1] + ", " +
          Colores.tiempos2manos_AI[2] + ", " + Colores.tiempos2manos_AI[3]);
        writer.WriteLine("Meñique izquierdo: ," + Colores.tiempos2manos_MI[0] + ", " + Colores.tiempos2manos_MI[1] + ", " +
          Colores.tiempos2manos_MI[2] + ", " + Colores.tiempos2manos_MI[3]);          }
    }
    if ((Application.loadedLevel == 4) && (Colores.tiempos2manos_PD[0] != 0) && (File.Exists(GetInput.ruta)))       {
      using (StreamWriter writer = File.AppendText(GetInput.ruta))          {
        writer.WriteLine(" ");
        writer.WriteLine("TIEMPOS DE LA MANO DERECHA DURANTE LA PRUEBA CON AMBAS MANOS");
        writer.WriteLine("Pulgar derecho: ," + Colores.tiempos2manos_PD[0] + ", " + Colores.tiempos2manos_PD[1] + ", " +
          Colores.tiempos2manos_PD[2] + ", " + Colores.tiempos2manos_PD[3]);
        writer.WriteLine("Indice derecho: ," + Colores.tiempos2manos_ID[0] + ", " + Colores.tiempos2manos_ID[1] + ", " +
          Colores.tiempos2manos_ID[2] + ", " + Colores.tiempos2manos_ID[3]);
```

```
            writer.WriteLine("Corazon derecho: ," + Colores.tiempos2manos_CD[0] + ", " + Colores.tiempos2manos_CD[1] + ", " +
              Colores.tiempos2manos_CD[2] + ", " + Colores.tiempos2manos_CD[3]);
            writer.WriteLine("Anular derecho: ," + Colores.tiempos2manos_AD[0] + ", " + Colores.tiempos2manos_AD[1] + ", " +
              Colores.tiempos2manos_AD[2] + ", " + Colores.tiempos2manos_AD[3]);
            writer.WriteLine("Meñique derecho: ," + Colores.tiempos2manos_MD[0] + ", " + Colores.tiempos2manos_MD[1] + ", " +
              Colores.tiempos2manos_MD[2] + ", " + Colores.tiempos2manos_MD[3]);          }
        }
        Application.Quit();    }
}
```

BarChart

Debido a la extensión del código se muestra en este anexo tan solo la parte correspondiente al dedo meñique de la mano izquierda, siendo necesario programar los mismos pasos para cada dedo.

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class BarChart : MonoBehaviour {
    public GameObject Barra_MI;
    public GameObject Barra_AI;
    public GameObject Barra_CI;
    public GameObject Barra_II;
    public GameObject Barra_PI;
    public GameObject Barra_PD;
    public GameObject Barra_ID;
    public GameObject Barra_CD;
    public GameObject Barra_AD;
    public GameObject Barra_MD;
    void Start () {
        int n_MI = 8;
        int n_AI = 8;
        int n_CI = 8;
        int n_II = 8;
        int n_PI = 8;
        int n_PD = 8;
        int n_ID = 8;
        int n_CD = 8;
        int n_AD = 8;
        int n_MD = 8;
        float suma_MI = 0;
        float suma_AI = 0;
        float suma_CI = 0;
        float suma_II = 0;
        float suma_PI = 0;
        float suma_PD = 0;
        float suma_ID = 0;
        float suma_CD = 0;
        float suma_AD = 0;
        float suma_MD = 0;
        for (int i = 0; i < 4; i++)       {
            suma_MI = suma_MI + Colores.tiempos1mano_MI[i];
            suma_MI = suma_MI + Colores.tiempos2manos_MI[i];

            if (Colores.tiempos1mano_MI[i] == 0)          {
                n_MI--;          }
            if (Colores.tiempos2manos_MI[i] == 0)          {
                n_MI--;          }
        }
        float promedio_MI = suma_MI / n_MI;
        Vector3 tamBarraMI = Barra_MI.transform.localScale;
        tamBarraMI.y = promedio_MI;
        Barra_MI.transform.localScale = tamBarraMI;
        Vector3 posBarraMI = Barra_MI.transform.position;
        posBarraMI.y = ((promedio_MI) / 2) - 3;
        Barra_MI.transform.position = posBarraMI;   }
}
```

TextOverButton

```
using UnityEngine;
using System.Collections;
using UnityEngine.EventSystems;
using UnityEngine.UI;

public class TextOverButton : MonoBehaviour, IPointerEnterHandler, IPointerExitHandler{
   public GameObject childText = null;

   void Start () {
      Text text = GetComponentInChildren<Text>();
      if (text != null)       {
         childText = text.gameObject;
         childText.SetActive(false);       }
   }

   public void OnPointerEnter(PointerEventData eventData)    {
      childText.SetActive(true);    }

   public void OnPointerExit(PointerEventData eventData)    {
      childText.SetActive(false);    }

}
```